Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Electrónica





# Plataforma de software empotrado para la implementación de algoritmos de audio y video en el DSP de la arquitectura OMAP-L138

**Informe de Proyecto de Graduación para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura**


**Esteban A. Zúñiga Mora**

**Jorge A. Hidalgo Chaves**



**Cartago, I semestre 2010**



Esteban A. Zúñiga Mora, Jorge A. Hidalgo Chaves, *Plataforma de software empotrado para la implementación de algoritmos de audio y video en el DSP de la arquitectura OMAP-L138* , IE-ITCR ©, I semestre 2010.

**INSTITUTO TECNOLOGICO DE COSTA RICA**
**ESCUELA DE INGENIERIA ELECTRONICA**
**PROYECTO DE GRADUACIÓN**
**TRIBUNAL EVALUADOR**
**ACTA DE EVALUACIÓN**

Proyecto de Graduación defendido ante el presente Tribunal Evaluador como requisito para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura, del Instituto Tecnológico de Costa Rica.

Miembros del tribunal

**MSEE Ing. Juan Scott Chaves**
Profesor lector

**Dr. Ing. Carlos Meza Benavides**
Profesor lector

**Dr. Ing. Pablo Alvarado Moya**
Profesor asesor

Los miembros de este Tribunal dan fe de que el presente trabajo de graduación ha sido aprobado y cumple con las normas establecidas por la Escuela de Ingeniería Electrónica

Cartago, I semestre 2010

**Declaramos que el presente proyecto de Graduación ha sido realizado enteramente por nosotros, utilizando y aplicando literatura referente al tema e introduciendo conocimientos propios.**

**En los casos en que hemos utilizado bibliografía, hemos procedido a indicar las fuentes mediante las respectivas citas bibliográficas.**

**En consecuencia, asumimos la responsabilidad total por el trabajo realizado y por el contenido del correspondiente informe final.**

Esteban Zúñiga Mora
Céd:1-1296-0855

Jorge Hidalgo Chaves
Céd:1-1289-0993

**Cartago, I semestre 2010**

# Resumen


Las demandas de espacio de almacenamiento y rendimiento para los sistemas empotrados destinados al procesamiento digital de señales (DSP) han impulsado el desarrollo de nuevas tecnologías enfocadas en el procesamiento de audio digital y procesamiento de gráficos de alta calidad, entre otras funciones. Dichos sistemas empotrados han logrado expandirse en el mercado mundial de los dispositivos electrónicos gracias a su amplio rango de aplicación.

Es por este motivo que fabricantes de circuitos electrónicos como Texas Instruments (TI) se han dado a la tarea de desarrollar dispositivos destinados específicamente al procesamiento digital de señales (DSP). RidgeRun, la empresa para la cual se desarrolla el proyecto descrito en esta tésis, provee soluciones de software para diversas plataformas que integran Sistemas en Chip (SoC) de la familia DaVinci™ y OMAP™ diseñados y producidos por Texas Instruments™ y se encuentra interesada en el desarrollo de herramientas de software basadas en la tecnología DSP.

En este trabajo se crea una herramienta destinada a la construcción de aplicaciones DSP. Se seleccionó la micro-arquitectura OMAP-L138, específicamente la tarjeta ZoomEvm como plataforma de desarrollo. Antes de la realización de este proyecto no existía un marco de trabajo que vinculara el conjunto de herramientas, ya existentes, para construir aplicaciones multimedia en la micro-arquitectura del Sistema en Chip (SoC) OMAP-L138 que aprovechen las capacidades de procesamiento digital de señales (DSP) disponibles para audio y video, a través de algoritmos diseñados especialmente para este tipo de dispositivos.

El conjunto de herramientas implementadas se basa en componentes de software libre, provistos por la comunidad GNU/Linux y adaptados a la tecnología OMAP por Texas Instruments. Los elementos son unificados en un kit de desarrollo de software empotrado, encargado de automatizar los procesos de construcción y compilación de componentes de software dentro de los cuales se destaca un gestor de arranque, el kernel de GNU/Linux y un sistema de archivos, así como como mecanismos de instalación del software en una tarjeta de desarrollo.


# Palabras claves


*Algoritmos DSP, Sistema en Chip OMAP- L138, Sistemas empotrados, Audio, Video, Software libre, GNU/Linux*

# Abstract

The demands of storage space and performance for embedded systems for digital signal processing (DSP) have driven the development of new technologies focused on digital audio processing and high-quality graphics processing, among other functions. These embedded systems have grown in the worldwide market for electronic devices thanks to its wide range of application.

It is for this reason that electronics manufacturers like Texas Instruments (TI) have been devoted to the task of developing devices intended specifically to digital signal processing (DSP). RidgeRun, the company where the project described in this thesis has been developed, provides software solutions for various platforms that integrate system-on-Chip (SoC) families DaVinci ™ and OMAP ™ designed and produced by Texas Instruments ™ and is interested in the development of software tools based on DSP technology.

With the aim of creating a framework reference based on the existing software tools for the construction of DSP applications the micro-architecture OMAP-L138, specifically the Zoom Evm card has been chosen as a development platform. Before this project a standard framework reference to build multimedia applications in the micro-architecture of the System on Chip (SoC) OMAP-L138 capable to exploit the features of digital signal processing (DSP) available for audio and video through algorithms specially designed for these devices was not available.

The toolkit implemented is based on open source components, provided by the GNU/Linux community and adapted by Texas Instruments to OMAP technology. The elements are unified in a software development kit in charge of automating the construction and compilation of software components which include a boot loader, the kernel of GNU/Linux and a file system and the mechanisms such as software installation on a development board.



# Keywords

*DSP Algorithms, System on Chip OMAP- L138, Embedded System, Audio, Video, Open Source Software, GNU/Linux*

# Dedicatoria

*Dedico este trabajo a mi familia por su paciencia y amor incondicional durante esta etapa en mi vida, gracias por preocuparse y apoyarme siempre en los momentos más difíciles.*
*En especial a mi padre y a mi madre, mis dos motivaciones ... mis dos razones para nunca rendirme.*

*Gracias.*

Esteban A. Zúñiga M

*"Our greatest weakness lies in giving up.*
*The most certain way to succeed is always to try just one more time."*

***Thomas A. Edison***

*Este proyecto culmina con una etapa más de mi vida. Deseo dedicar, pero sobre todo agradecer el apoyo incondicional de mi familia, su paciencia y el amor que siempre han brindado en mi vida, para nunca rendirme y seguir mis sueños.*
*En especial a mi padre y a mi madre, dos pilares que han sembrado en mí actitudes de las cuales me siento orgulloso. Su apoyo y amor nunca han faltado.*

*Gracias.*

Jorge A. Hidalgo Chaves

*Lo que sabemos es una gota de agua; lo que ignoramos es el océano.*

***Sir Issac Newton***

# Agradecidimiento

*Primeramente queremos dar gracias a Dios, por acompañarnos en esta etapa de nuestras vidas y darnos la fortaleza para nunca rendirnos y acabar con grandes éxitos.*
*A la empresa RidgeRun, por abrirnos las puertas para poder realizar nuestro proyecto de graduación, y convertirse en una familia para nosotros.*
*A nuestro asesor, Dr. Pablo Alvarado M., por confiar en nuestra capacidad y aconsejarnos cuando más lo necesitamos.*
*Al Ing. Diego Dompe, por su dedicación y empeño, por ayudarnos a salir adelante durante las etapas críticas de nuestro proyecto.*
*A todos nuestros compañeros que forman parte de la familia RidgeRun, por hacernos sentir bienvenidos . Cada uno de ellos realizó aportes importantes a nuestro proyecto, que nos ayudaron a crecer profesional y personalmente.*

*Gracias.*

# Índice general

# Índice de figuras

# Índice de tablas

# Índice de abreviaturas

**API** Application Programming Interface

**ALSA** Advance Linux Sound Architecture

**ADC** Analog to Digital Convertor

**CCS** Code Composer Studio

**CE** Codec Engine

**CS** Codec Server

**CPU** Central Processing Unit

**DAC** Digital to Analog Convertor

**DMA** Direct Memory Access

**DMAI** Davinci Media Aplication Interface

**DVSDK** Digital Video Software Development Kit

**DSP** Digital Signal Processing

**GCC** GNU Compiler Collection

**GPP** General Porpuse Proccesor

**IPC** Inter Processor Comunication

**JFF2** Journalling Flash File System version 2

**I2C** Inter-Integrate-Circuit

**LAN** Local Area Network

**LCD** Liquid Crystal Display

**McASP** Multichannel Audio Serial Port

**MMU** Memory Management Unit

**NFS** Network File Sytem

**OMAP** Open Multimedia Application Platform

**OHCI** Open Host Controller Interface

**OTG** On-The-Go

**RPC** Remote Procedure Call

**RTSC** Real-Time Software Components

**SDK** Software Development Kit

**SDRAM** Synchronous Dynamic Random Access memory

**SoC** System On Chip

**TI** Texas Instruments

**TICGT** Texas Instruments Code Generation Tools

**UART** Universal asynchronous receiver/transmitter

**VISA** Video , Image , Speech and Audio

**XDAIS** eXpressDsp Algorithm Interoperability Standard

**xDM** eXpressDSP Digital Media

# 1. Introducción

## 1.1. Los Sistemas Embebidos

La tecnología electrónica está asociada a la utilización de dispositivos y sistemas computacionales tales como circuitos integrados, microprocesadores y redes de comunicación, entre otros. Estos se orientan al manejo de información y control de procesos que permiten incrementar la productividad de las empresas, la seguridad de los equipos y el ahorro energético entre otros con el fin de mejorar la calidad de vida [4].

Un sistema empotrado es un dispositivo computacional, producto de la combinación de elementos de *software* y *hardware* con el fin de realizar aplicaciones de propósito específico. Se pueden enfocar de una manera especializada para ejecutar tareas en tiempo real y además pueden ser configurados para minimizar su consumo de energía dependiendo de la función para la cual son diseñados [9].

Algunas aplicaciones en donde se utilizan sistemas empotrados son [6]:

- La televisión digital
- Automóviles
- Medicina
- Sistemas de control industrial
- Redes
- Telefonía

El mercado de los sistemas embebidos involucra varias etapas, desde el diseño hasta la fabricación y venta del producto. Es dentro de la etapa del desarrollo de un sistema empotrado que interactúan varias compañías encargadas cada una de tareas como la elaboración de los circuitos integrados, diseño e implementación de sensores y tarjetas de evaluación, y finalmente el desarrollo de software para el lanzamiento y mantenimiento de un producto al mercado [6].

La tabla 1.1 muestra áreas para las cuales se desarrollan sistemas embebidos, así como ejemplos en cada una de ellas.

Tabla 1.1.: Áreas de desarrollo de los sistemas embebidos

| **Mercado** | **Dispositivo** |
|---|---|
| Automóviles | Inyección electrónica<br>Frenos<br>Control de Vidrios<br>Control de Asientos |
| Consumo | Televisión<br>DVD<br>PDA<br>Electrodomésticos<br>Celulares<br>Cámaras<br>Reproductores Multmedia y GPS |
| Control industrial | Sistemas de Robótica y de Control |
| Medicina | Bombas de transfusión<br>Cámaras de alta definición |
| Redes | Routers<br>Gateways<br>Hube |
| Oficinas | Fax<br>Fotocopiadoras<br>Impresoras<br>Monitores |

Los sistemas operativos basados en el *kernel* de GNU/Linux se han introducido con solidez en el desarrollo de sistemas empotrados y a la fecha se estima que el 40 % de los sistemas empotrados existentes en el mercado utilizan GNU/Linux empotrado [3]. El interés principal al utilizar ARM/Linux, como también se le conoce, radica en los siguientes puntos[3]:

- Libre de costos de regalías en tiempo de ejecución
- Variedad de herramientas que pueden ser descargadas libremente (bibliotecas, compiladores, archivos de comandos)
- Mayor disponibilidad de controladores de dispositivos, en comparación con los sistemas operativos propietarios
- Disponibilidad y control sobre el código fuente
- Disponibilidad de proyectos de código abierto diseñados para GNU/Linux además de un apoyo permanente de la comunidad de código abierto

## 1.2. Algoritmos de DSP y su relación con la arquitectura OMAP-L138

La tarjeta *OMAP-L138 ZoomEvm* es fabricada por *Logic PD* [14]. Este dispositivo utiliza un *SoC OMAP-L138* que integra un microprocesador *ARM 9* y un procesador *DSP* con capacidades de punto flotante de la familia *C674X,* basados en la tecnolgía DaVinci de Texas Instruments para procesamiento multimedia. El dispositivo embebido posee una serie de periféricos que permiten el desarrollo de aplicaciones multimedia, tales como puertos de entrada y salidas de audio y video, puerto Ethernet, puertos USB, decodificadores de audio y decodificadores de video.

En la figura 1.1 se observa un modelo de la ZoomEvm acompañada de su tarjeta de expansión. Asímismo puede añadirse como hardware adicional a la tarjeta una pantalla táctil con dimensiones de 480 x 272 pixeles.



Figura 1.1.: Zoom EVM, tarjeta producida por LogicPD

Al ser esta tarjeta una plataforma que incluye un procesador DSP, su uso y funcionalidad pueden ser explotados para el desarrollo de algoritmos específicos a este tipo de aplicaciones. Estos algoritmos son utilizados principalmente para construir aplicaciones orientadas al procesamiento de audio, video, imágenes y voz. La realización de este proyecto se centra en la integración para el desarrollo de algoritmos destinados a procesar audio y video.

El rendimiento del sistema como un conjunto mejora cuando los procesadores ARM y DSP ejecutan tareas en paralelo, brindando al usuario la posibilidad de acceder a un sistema donde los procesos y los servicios pueden coexistir en conjunto sin saturar las capacidades del dispositivo. Dado que ejecutar diversas aplicaciones concurrentes únicamente en el procesador ARM conlleva a una saturación del sistema, es posible afirmar que la realización del proyecto busca solucionar esta problemática.

Antes de realizar este proyecto, no existía un marco de trabajo que vinculara el conjunto de herramientas existentes y permitiese la construcción de aplicaciones multimedia para la micro-arquitectura del *SoC OMAP-L138*. Esto restringe el aprovechamiento de las capacidades de procesamiento digital de señales (DSP) disponibles en el dispositivo para crear y ejecutar algoritmos DSP para procesamiento de audio y video.

Al inicio del desarrollo con la tarjeta *OMAP-L138 ZoomEvm,* el soporte oficial del fabricante solamente incluye una imagen precargada del gestor de arranque (u-boot) y una imagen del *kernel* de GNU/Linux en su sistema. Con respecto a la perspectiva de un desarrollador, la tarjeta presenta software cuya configuración es desconocida y no necesariamente adecuada para lo que se requería implementar en el proyecto actual.

Dentro de este entorno de software, el desarrollo de aplicaciones enfocadas al uso de algoritmos dedicados a DSP (basados en tecnología de Texas Instruments TMS320C6000) implica dominio de conocimento detallado acerca del software y hardware de la tarjeta. Debe invertirse una cantidad de tiempo considerable para lograr tal objetivo, tiempo valioso que los clientes o los usuarios no desean utilizar por razones de producción, tiempo de lanzamiento al mercado o falta de personal capacitado para dicha tarea.

La solución propuesta es un kit de desarrollo de software empotrado "***Software Development Kit* (SDK)**", con una estructura similar a los que RidgeRun ha implementado para otras plataformas a las cuales da soporte (*Dm-6446, Dm-355, Dm-365, OMAP-3x, etc.*). Específicamente, el proyecto se enfoca en crear un marco de trabajo que incorpore herramientas encargadas de dar soporte a algoritmos de DSP, en conjunto con las herramientas que Texas Instruments provee para la creación y ejecución de aplicaciones que utilicen este tipo de algoritmos.

La figura 1.2 muestra la estructura de bloques que representa la solución planteada. Cada uno de los bloques representa una parte del SDK a implementar.

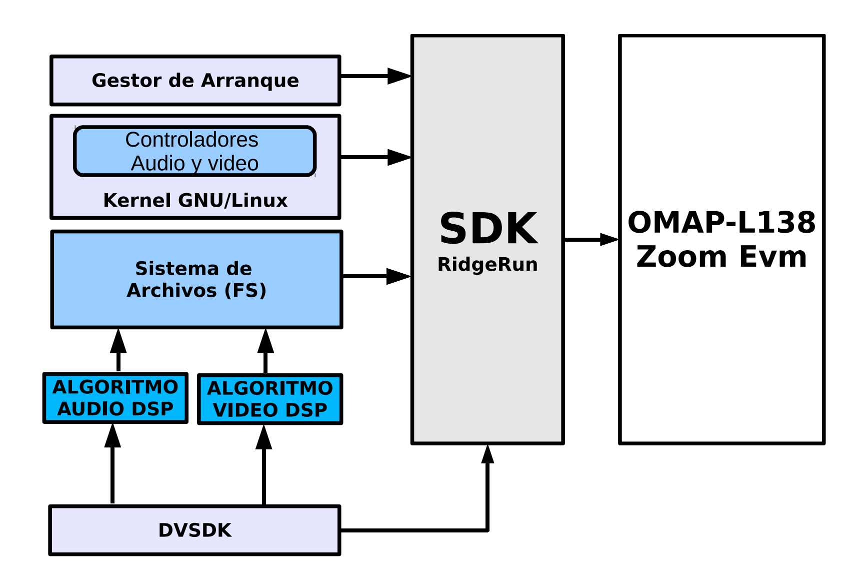



Figura 1.2.: Diagrama generalizado de construcción del SDK

El objetivo del proyecto es que un usuario pueda utilizar el *SDK* resultante para la arquitectura de la *OMAP-L138 ZoomEvm* y pueda agregar sus aplicaciones, especialmente para DSP, sin preocuparse por controladores de hardware u otros inconvenientes de firmware en la arquitectura. La integración de las herramientas tiene como base un conjunto de secuencias de comandos (**scripts**) que incluyen e integran los módulos provistos por Texas Instruments™, adaptados a la arquitectura *OMAP-L138*.

El *SDK* está compuesto por una serie de componentes tales como un gestor de arranque, una versión del *kernel de GNU/Linux*, un compilador cruzado (gcc para el procesador *ARM*), un sistema de archivos, un instalador, el *DVSDK* y bibliotecas de código abierto que han sido integradas anteriormente para otras plataformas. Dentro de estas bibliotecas se encuentran *Qt*, *GStreamer* y *D-Bus*.

El *DVSDK* incluye herramientas de software embebido, propiedad de Texas Instruments, las cuales se utilizan para el desarrollo de aplicaciones multimedia tanto para el procesador *ARM* como para la construcción de algoritmos de DSP. Este paquete está integrado por componentes especializados y enlazados entre sí:

- *DSPLink*
- *DPSBios*
- *CodecEngine*
- *TI CodeGen Tools*
- *XDAIS*
- *Dmai*
- *XdcTools*

◦ *CodecServer*

El paquete *DVSDK* adaptado para la arquitectura *OMAP-L138* permite la construcción automatizada de los componentes mencionados anteriormente, para que el usuario solamente dirija su atención a la construcción del algoritmo y la aplicación encargada de ejecutarlo.

El enfoque de la solución actual es crear una infraestructura de software que permita construir algoritmos basados en el procesamiento digital de señales en una arquitectura como la *OMAP-L138*. El desarrollo de aplicaciones de este tipo requiere de la integración de un canal de comunicación entre los procesadores ARM y DSP que permita un flujo bi-direccional de datos entre ambos. Así, la aplicación que hace uso del algoritmo se ejecuta en el procesador ARM mientras que el algoritmo de procesamiento de datos multimedia se ejecuta en el DSP.

Como parte de la solución, se desarrollaron tanto un algoritmo de procesamiento de video (conversión de espacio de colores) como un algoritmo de audio (filtro FIR) que ejemplifican el uso de la *API* de *iUniversal* [34] (extensión de la *API XDAIS* de TI) para la creación de algoritmos DSP estándar[1]. Finalmente, la ejecución de los algoritmos se efectuó por medio de una aplicación basada en *GStreamer*. Los algoritmos se encapsulan con elementos de *GStreamer* en un canal de comunicación estándar, demostrando las capacidades del DSP para procesamiento matemático y en general la utilidad del marco de integración generado por el SDK.

## 1.3. Objetivos y Estructura del Trabajo

Este trabajo ha tenido como objetivo implementar una plataforma de desarrollo de software (SDK) para la arquitectura OMAP-L138, que sirve como base para la integración de algoritmos de audio y video enfocados a la arquitectura del DSP. Para este propósito es necesario implementar e instalar en la tarjeta un gestor de arranque de código abierto, un sistema operativo y un sistema de archivos (*file system*) como ambiente base para control y comunicación con el hardware.

Con la integración del sistema operativo *GNU/Linux* se habilitan los módulos controladores de los dispositivos de reproducción de audio y video de la tarjeta en uso. Una vez hecho lo anterior, se integran en el *SDK* los paquetes o interfaces provistos por Texas Instruments para establecer la comunicación entre el DSP y el ARM en la tarjeta, y también se crea el

[1] Existen otras APIs conocidas como VISA, que son también extensiones de *XDAIS*, dedicadas a la construcción de algoritmos decodificares y encodifcadores de audio, video e imágenes.

marco de referencia para la construcción de un algoritmo de DSP con la interfaz *iUniversal* que suministra el paquete *CodecEngine*.

Una forma de comprobar la funcionalidad de las herramientas integradas para la construcción y ejecución de algoritmos es la creación de un algoritmo de audio y uno de video, siguiendo los estándares de *XDAIS* e integrarlos en una aplicación de *GStreamer* para demostrar las capacidades del hardware y del software.

El capítulo 2 describe la estructura de un *SDK* de RidgeRun así como de los paquetes que Texas Instruments provee; estos paquetes están descritos en la sección 2.2.6. Además, en este capítulo se dedica una sección a la teoría básica de espacios de colores y filtros tipo FIR, que son los modelos implementados para ejemplificar la funcionalidad de la herramienta final.

Los capítulos 3, 4 y 5 detallan la implementación realizada de los algoritmos, su integración con el marco de *iUniversal* y el uso de las herramientas añadidas en el *SDK* para ejecutar algoritmos de DSP a través de una aplicación de *GStreamer*.

En el capítulo 6 se muestran las pruebas de rendimiento efectuadas a los algoritmos creados con el marco de herramientas que ofrece el *SDK*. Se muestra una serie de resultados de consumo de CPU que comparan el sistema durante la ejecución de una aplicación que usa el procesador DSP y otra que solamente se ejecuta en el procesador ARM. Se discuten los resultados de las pruebas tanto para audio como para video y se efectúa el análisis respectivo.

Finalmente el capítulo 7 resume el trabajo efectuado en una serie de conclusiones referentes al rendimiento de las aplicaciones cuando hacen uso del DSP y cuando solamente se ejecutan en el ARM . El capítulo finaliza con recomendaciones alrededor de las limitaciones experimentadas durante el desarrollo del proyecto y se explora su solución para trabajos futuros.

# 2. Marco Teórico

## 2.1. Arquitectura OMAP-L138 y la Zoom Evm

La tarjeta *OMAP-L138 ZoomEvm* es un kit de desarrollo utilizado para evaluar la funcionalidad de la arquitectura *OMAP-L138*. Está dotada de un procesador *ARM926EJ-S* que funciona en paralelo con un procesador *DSP TMS320C6748*; sin embargo el DSP actúa como un periférico esclavo del ARM.

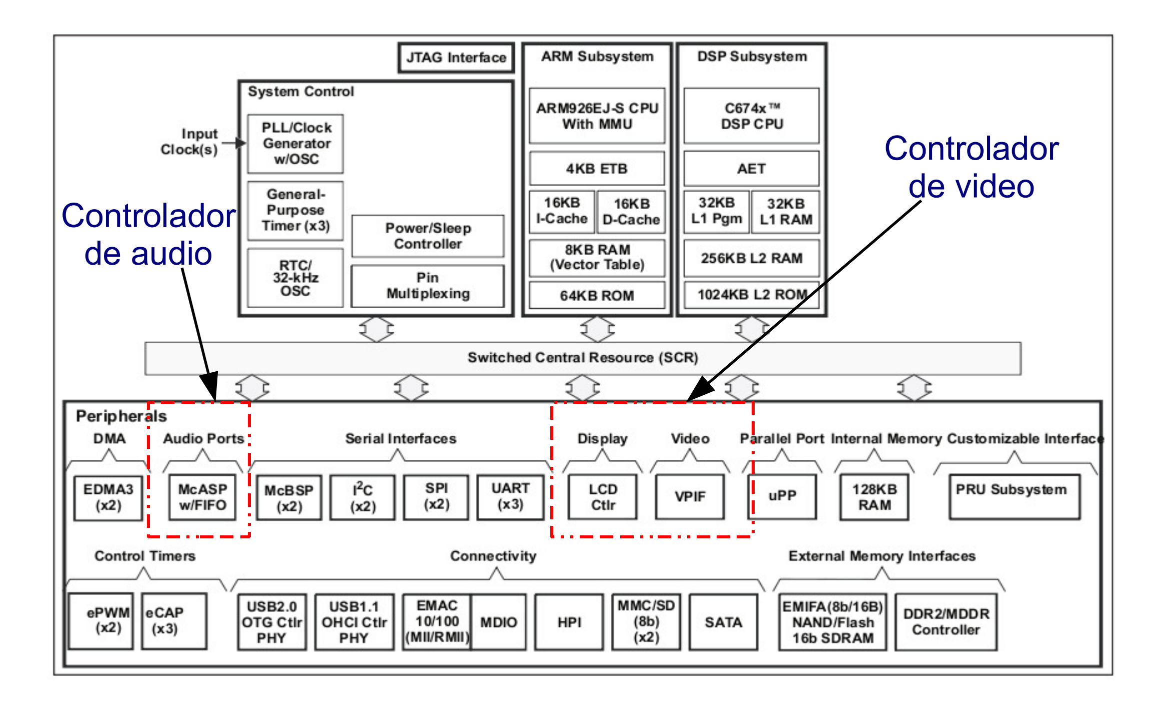



Figura 2.1.: Diagrama funcional de bloques de la arquitectura OMAP-L138. (Tomado de [47])

El núcleo ARM tiene una arquitectura segmentada (pipeline) lo que permite al procesador y a la memoria del sistema operar continuamente. El *OMAP-L138* está destinado al desarrollo de aplicaciones basadas en procesamiento digital de señales y se destaca en el sector industrial

para desarrollo de dispositivos de audio, aplicaciones médicas y telecomunicaciones, entre otros [47].

La tarjeta proporciona un entorno de desarrollo de software para procesamiento digital de señales e incluye periféricos especialmente seleccionados para aprovechar al máximo las capacidades que ofrece la arquitectura. El diagrama de bloques provisto por el fabricante se muestra en la figura 2.1, en donde se resaltan los módulos de hardware que se utilizan en este proyecto para el despliegue de video (controlador de *LCD*) y la reproducción de audio (controlador *McASP*). Estos módulos de hardware pueden configurarse para ser utilizados a través de las opciones de construcción del kernel de *GNU/Linux*, con el objetivo de habilitar los periféricos respectivos de la tarjeta *Zoom Evm*.

La figura 2.2 muestra el diagrama de bloques del controlador de LCD. Éste se compone de dos controladores independientes: el controlador "Raster" y el controlador de la interfaz de pantalla (LIDD). Cada bloque funciona de manera independiente y sólo uno de ellos está activo a la vez. El camino que sigue el flujo de datos de video está resaltado con líneas gruesas.

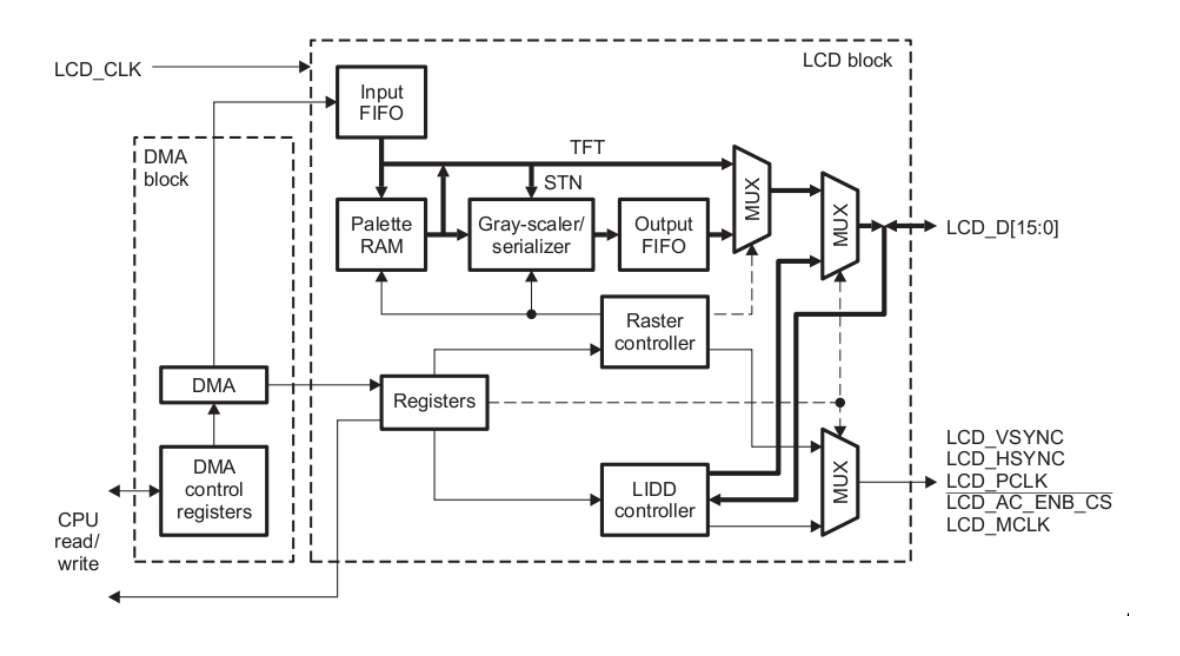



Figura 2.2.: Diagrama del Controlador de Despliegue de Video LCD. (Tomado de [38])

Estos controladores son responsables de administrar apropiadamente el despliegue de datos. El procesador ARM está acompañado de un dispositivo de DMA (Acceso Directo a Memoria) que proporciona un flujo constante de datos de la memoria a la pantalla LCD

externa, a través de los controladores. También existe acceso del CPU para leer y escribir los registros.

Con respecto al dispositivo de audio multi-canal serial (*McASP*), éste funciona como un puerto de audio de propósito general. Dicho dispositivo está optimizado para las necesidades de las aplicaciones de audio. El *McASP* es útil para formatos de transmisión como: multiplexación por división de tiempo (*TDM*), el protocolo "*Inter-IC Sound 2*" (*I2S*) y la interfaz de transmisión digital intercomponente de audio (*DIT*).

El McASP transmite y recibe secciones de datos que pueden operar en forma sincronizada, o de forma independiente con relojes separados por medio de diferentes modos de transmisión con diferentes formatos para el flujo de éstos. En la figura 2.3 se muestra un diagrama completo de la estructura que compone el *McASP* integrado en la arquitecura *OMAP-L138*.

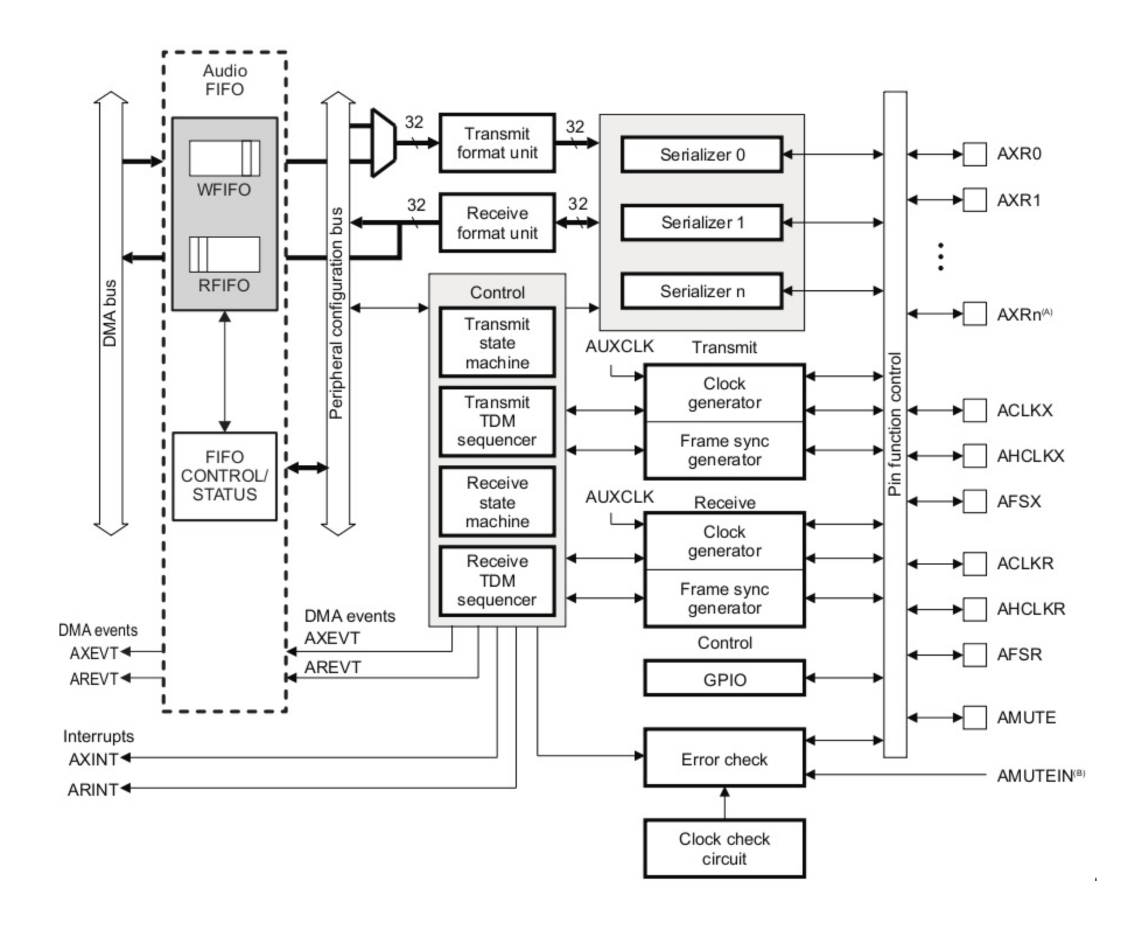



Figura 2.3.: Diagrama de bloques del controlador McASP. (Tomado de [39])

## 2.2. El *SDK* de *RidgeRun*

La construcción de *software* para sistemas embebidos se traduce en una tarea de compilación especializada, es decir que cada paquete debe de ser compilado para una plataforma en específico a través de un proceso específico [26]. El *SDK* de *RidgeRun* fue diseñado con el fin de simplificar el proceso de construcción de aplicaciones de *software* empotrado en las plataformas para las que se ofrece soporte.

La construcción del paquete de *software* tiene como base un conjunto de secuencias de comandos que construyen un kit de de desarrollo compatible tanto con la aquitectura a utilizar como con la tarjeta correspondiente [24]. El *SDK* de RidgeRun es una herramienta destinada al desarrollo de aplicaciones de alto nivel, ya que brinda al usuario la capacidad de desarrollar sus aplicaciones sin tener que preocuparse por la estructura de capas inferiores de *software* o de *hardware*. Las referencias al *hardware* dentro del *SDK* de RidgeRun se acceden de una forma estructurada, y faculta al usuario la posibilidad de referirse a él durante el desarrollo de una aplicación.

### 2.2.1. Características del *SDK* de RidgeRun

Adaptar *GNU/Linux* a cualquier plaforma de *hardware* nueva es una tarea que implica conocimiento avanzado de sistemas computacionales [11]. Sin embargo el *SDK* de RidgeRun es un kit de desarrollo para Linux Empotrado que está diseñado para simplificar ese trabajo. En él se integran y utilizan herramientas de código abierto provistas por Texas Instruments y la comunidad de Código Abierto. Actualmente existen SDK optimizados para diversas plataformas, dentro de las cuales se destacan la *OMAP*35*x*, la *DM*365 y la *DM*355.

Dentro de las utilidades que el *SDK* incluye se encuentran paquetes como el gestor de arranque, una versión del kernel *GNU/Linux* y aplicaciones y bibliotecas tales como *GStreamer*, *Qt*, *D-Bus* entre otros [24].

La creación de un *SDK* [24] está basado en un conjunto de programas y secuencias de comandos tales como :

- El sistema de GNU/Make [23]
- Archivos de comandos de línea (script de shell), utilizados para automatizar tareas
- Un conjunto de Makefiles personalizados para cada aplicación y utilizado para integrar:
  - Un gestor de arranque (bootloader)
  - Un núcleo de Linux, basado en kconfig
  - Un sistema de archivos (file system)
  - Una cadena de herramientas (toolchain) que permite compilar los paquetes de software para la arquitectura deseada

    - Una configuración propia de cada arquitectura y tarjeta
    - El DVSDK, dedicado a aplicaciones de multimedia

- Detección e instalación de las herramientas necesarias para compilar:
    - El gestor de arranque
    - El kernel GNU/Linux
    - El sistema de archivos
    - Las bibliotecas de GNU/Linux
    - Las aplicaciones extra

El SDK de RidgeRun automatiza el proceso de compilación, así como la instalación de los componentes de *software* en la tarjeta. Éstos pueden ser configurados mediante un menú principal con sub-menús de configuración por componente, tal como muestra la figura 2.4.

Figura 2.4.: Menú principal de configuración del SDK

A continuación se explican detalladamente los bloques escenciales que conforman la herramienta de desarrollo provista por *RidgeRun*.

### 2.2.2. Gestor de arranque (*Bootloader*)

*GNU/Linux* no puede iniciar sin un código especializado para cada tarjeta que permita arrancar el sistema. Se requiere un gestor de arranque para configurar la memoria del sistema y copiar la imagen del *kernel* de un medio de almacenamiento persistente a una dirección de memoria de arranque de sistema. Una vez que se completa esta fase, el gestor de arranque ya no es necesario y le transfiere el control al *kernel* [11].

La característica principal del gestor de arranque es que se encuentra almacenado en una memoria no volátil; cada vez que se inicializa la tarjeta, el gestor de arranque es cargado automáticamente desde posiciones predeterminadas en la memoria de almacenamiento, llamadas vectores de arranque. Normalmente éste reside en una unidad de almacenamiento de tipo *NOR*, o una tipo *NAND*. La mayoría de las tarjetas *OMAP* utilizan *Das U-Boot* (Universal Boot Loader) [48] como gestor de arranque.

El gestor de arranque carga una aplicación por línea de comandos a través de una terminal serial que representa la comunicación entre la tarjeta y la computadora como se muestra en la figura 2.5. Este intercambio de información se realiza mediante herramientas como Minicom, TeraTerm y otros que interpretan la información que la tarjeta envía al puerto serial como secuencias de caracteres ASCII.

```
UBL customized by RidgeRun
Booting TI User Boot Loader
        UBL Version: 1.65
        UBL Flashtype: SPI
Starting SPI Memory Copy...
Valid magicnum, 0x55424CBB, found at offset 0x00010000.
   DONE
Jumping to entry point at 0xC1080000.


U-Boot customized by RidgeRun
U-Boot 2009.01 (Feb 16 2010 - 14:16:33)

DRAM:  64 MB
In:    serial
Out:   serial
Err:   serial
ARM Clock : 300000000 Hz
DDR Clock : 300000000 Hz
Ethernet PHY: GENERIC @ 0x00
Hit any key to stop autoboot:  0
OMAP-L138 EVM #
OMAP-L138 EVM #
OMAP-L138 EVM #
OMAP-L138 EVM #
OMAP-L138 EVM #
OMAP-L138 EVM #
```

Figura 2.5.: Gestor de Arranque de *Ridgerun* para la OMAP-L138

### 2.2.3. Núcleo o kernel de *GNU/Linux*

El rol principal del *kernel GNU/Linux* en el sistema es proveer una interfaz consistente entre las aplicaciones y el *hardware*. Este recurso aísla los programas de las capas de *hardware* y administra el *tiempo del procesador* (calendarización) que los programas utilizan. El *kernel* trata los programas como procesos: cada programa es independiente y solamente puede hacer peticiones de uso de *hardware,* sin accederlo directamente. Dicho proceso de acceso al *hardware* se muestra en la figura 2.6.

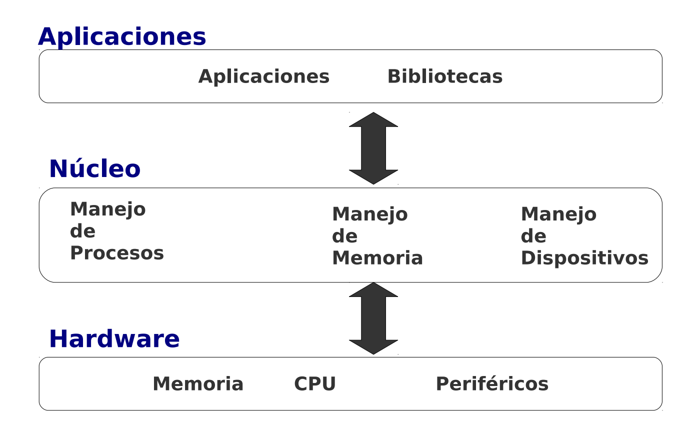



Figura 2.6.: Abstracción de capas esquematizada

El kernel de *GNU/Linux* se ocupa adicionalmente de gestionar los recursos de la memoria del sistema, decide cuándo, y qué proceso accede al sistema de archivos, controla el acceso a los periféricos y negocia los paquetes a través de la red, entre otras funciones. Esto permite a los programas evitar la implementación de esas capas.

Gracias al sistema *kconfig* presente en el *SDK* de *RidgeRun* para modificar las características del kernel, basta con habiltar las funciones en el menú de configuración y re-compilar el código fuente para disponer de los módulos necesarios. Véase la figura 2.7.

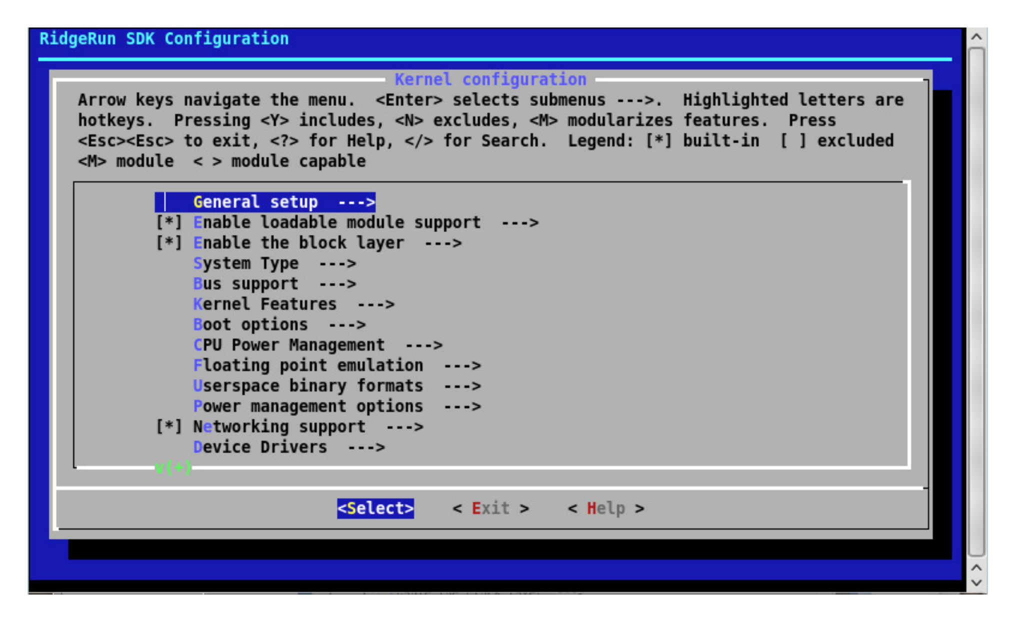



Figura 2.7.: Menú de configuración para el Kernel de Linux

### 2.2.4. Cadena de Herramientas (ToolChain)

Es un conjunto de herramientas de desarrollo que se utilizan para crear una aplicación o

programa y representan diferentes etapas en la obtención de código máquina ejecutable para un determinado procesador. Un conjunto de herramientas involucra: un editor de código fuente, un compilador y un enlazador (linker) para transformar código fuente en código ejecutable, bibliotecas para proveer interfaces, un sistema operativo, y un depurador.

El *GNU/Toolchain* es uno de los tantos paquetes de código abierto que realizan las funciones anteriormente mencionadas y se describe dentro de los siguientes apartados.

#### 2.2.4.1. Compilador GNU

Un compilador es un programa que permite transformar código fuente, escrito en un lenguaje de programación de alto nivel (C, C++, etc) en código objeto que pueda ser interpretado por el procesador [24]. En la figura 2.8 se puede observar el flujo de trabajo del compilador con respecto a un archivo de código fuente.

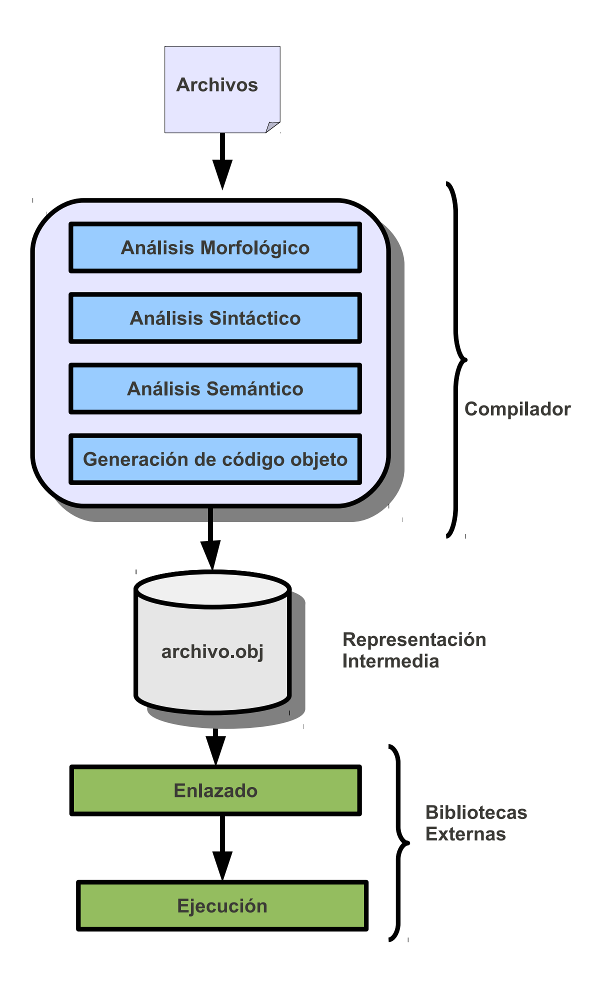



Figura 2.8.: Proceso de Compilación

Normalemente las operaciones que realiza un compilador son:

- Análisis morfológico, donde el compilador se encarga de crear una representación intermedia del programa fuente
- Análisis sintáctico, que contruye un modelo abstracto del código fuente
- Análisis semántico, que agrega al modelo abstracto reglas para la producción de código objeto
- Generación de código objeto, donde se genera el archivo ejecutable, posiblemente optimizado

Cuando se trabaja con un sistema embebido, usualmente el *software* o las aplicaciones se desarrollan y se crean en otra arquitectura, por ejemplo en una computadora de escritorio. Para el desarrollo de software embebido se necesita un compilador que sea capaz de crear código objeto para el sistema en cuestión. A esto se le denomina desarrollo cruzado o compilación cruzada.

El compilador GNU tiene la capacidad de realizar esta función para diferentes arquitecturas, como es el caso de este proyecto donde se utiliza un procesador ARM. No solo soporta código C, sino que también traduce código C++, FORTRAN, ADA y JAVA a lenguaje de máquina, entre otros [15].

Entre otras arquitecturas soportadas se pueden mencionar:

- PowerPC
- SPARC
- SuperH
- x86

#### 2.2.4.2. Herramientas binarias GNU (BinUtils)

Son una colección de herramientas de programación para la manipulación de código objeto. Dentro de estas utilidades existen dos herrramientas [15]:

1. as: Transforma código ensamblador, generado por gcc en un binario.
2. ld: El enlazador que genera el código objeto y lo interpreta en una biblioteca o en un archivo ejecutable.

#### 2.2.4.3. Depurador GNU

El depurador también es parte de la cadena de herramientas, como un depurador cruzado[15]. Se utiliza para depurar y optimizar aplicaciones que se ejecutan en el equipo de destino

(sistema embebido). En el mundo de de los sistemas embebidos un depurador muy usado es GDB.

### 2.2.5. Sistema de archivos

El sistema de archivos es una estructura de almacenamiento de la información como una colección de archivos. Formalmente, un sistema de archivos es un conjunto de tipos de datos abstractos que son implementados para el almacenamiento, la organización jerárquica, la manipulación, el acceso, el direccionamiento y la recuperación de datos [24].

Los sistemas de archivos almacenan datos acerca de los mismos archivos, conocidos como meta-información (metainfo).

Dentro de los datos que se almacenan se pueden citar:

- Nombre del archivo
- Fechas de creación
- Última modificación
- Tamaño del archivo
- Permisos del archivo

En los sistemas embebidos existen distintos tipos de sistemas de archivos utilizados con frecuencia, dentro de los cuales destacan el *NFS, el JFF2 y el EXT2*. A continuación se describe el *NFS*.

**NFS** (*Network File System*): Es un sistema de archivos compartido sobre Redes de Área Local (*LAN*). Este tipo de sistema de archivos posibilita que distintos sistemas conectados a una misma red accedan a archivos remotos como si se encontraran localmente. Dentro de sus principales ventajas destaca que no es necesario un puerto *RS232* para accederlo desde una computadora de escritorio. Existen diversos tipos de sistemas de archivos NFS tales como *NFSv2*, *NFSv3*, *NFSv4* y *WebNFS*.

### 2.2.6. DVSDK

El paquete de desarrollo de *software* de video digital (*DVSDK*) forma parte del contenido de las herramientas licenciadas por Texas Instruments que se integran en un *SDK* de *RidgeRun*. Este paquete contiene código basado en interfaces de programación de aplicaciones (*API*) para las tecnologías *DaVinci* y *OMAP*. Estas interfaces incluyen el soporte para diferentes plataformas de *hardware*, y permiten crear aplicaciones multimedia que pueden ser ejecutadas en un sistema embebido compatible con las tecnologías mencionadas.

Las herramientas incluyen paquetes de compilación, depuración, así como notas técnicas y documentación de apoyo. El código fuente es de propiedad libre, por lo que puede ser modificado a la necesidad del usuario. El *DVSDK* permite integrar aplicaciones tanto para el procesador *DSP* como para el procesador *ARM* [34]. Los paquetes más relevantes que forman parte de un DVSDK se pueden observar en la tabla 2.1 y se explicarán con más detalle en la sección 2.3.

Tabla 2.1.: Componentes del DVSDK

| Componente del DVSDK | Sección correspondiente |
|---|---|
| DSP/BIOS | 2.3.1 |
| CMEM | 2.3.2 |
| TI CodeGen Tools | 2.3.3 |
| CodecServer | 2.3.4 |
| XDAIS | 2.3.6 |
| CodecEngine | 2.3.7 |
| DMAI | 2.3.8 |

## 2.3. Herramientas de software de Texas Instruments

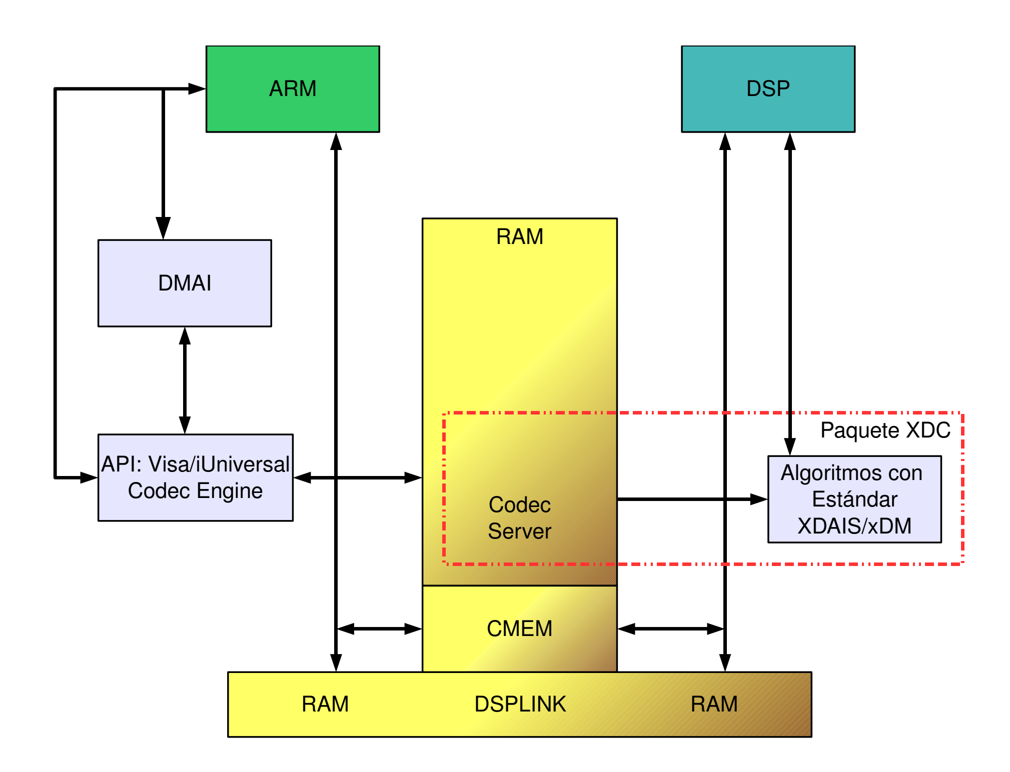



Figura 2.9.: Estructura de las herramientas de TI para el desarrollo de algoritmos de DSP

La integración de las herramientas provistas por TI en el SDK de la arquitectura OMAP-L138 permite el desarrollo de aplicaciones y algoritmos multimedia, tanto para el procesador

ARM como para el DSP. Esta integración se extiende desde el espacio de la aplicación de usuario y del algoritmo, hasta las capas de programación de *hardware* (*ARM o DSP*). En la figura 2.9, se muestra la estructura de software que se puede implementar para la ejecución de algoritmos de DSP en una arquitectura dual.

Antes de la realización de este proyecto, no existía un marco de trabajo que incluyera el conjunto de paquetes de software provistos por Texas Instruments y compatibles para la arquitectura OMAP-L138. Para lograr vincular los paquetes de software como se muestra en la figura 2.9, los paquetes deben ser configurados para el Sistema en Chip de TI que se desea utilizar. A continuación se presenta una descripción de las herramientas y la importancia de integrar y adaptar cada una de ellas con el SoC utilizado para el proyecto.

### 2.3.1. DSP/BIOS

Este paquete incluye el núcleo de un sistema operativo en tiempo real, escalable para el *DSP*. Diseñado específicamente por *Texas Instruments* para la familia de procesadores *TMS320* (serie de plataformas con *DSP* habilitados). Presenta la posibilidad de ser integrado con la arquitectura por medio del entorno de desarrollo *Code Composer Studio* [43].

Además, éste provee una serie de servicios dedicados al desarrollo de aplicaciones que no comprometen los plazos de tiempo de ejecución de procesos para un sistema en tiempo real, como por ejemplo: multi-hilos preemptivos, abstracción de *hardware* y análisis de señales en tiempo real. No requiere derechos de licencia en tiempo de ejecución e incluye soporte profesional de parte de TI [43].

Los servicios que provee este núcleo pueden ser invocados por aplicaciones que utilizan funciones en lenguaje C o en lenguaje ensamblador, y está diseñado para implementar la comunicación entre el sistema anfitrión (host) y el objetivo (target) [43], como se muestra en la figura 2.10. Algunos de los servicios de este núcleo se listan en la tabla 2.2.

Con este marco de trabajo, en el lado del anfitrión (host) se construyen los programas que ejecutan la *API* de *DSP/BIOS* (a través de código C o ensamblador). Las herramientas de configuración permiten definir objetos que pueden utilizarse en los programas que se crean. Lo siguiente es compilar y enlazar el programa.

Tabla 2.2.: Servicios que ofrece DSP/BIOS

| Funciones | Descripción |
|---|---|
| Interrupciones de hardware | Interfaz de interrupición de hardware en el núcleo de DSP/BIOS |
| Interrupciones de Software | Hilos preemptivos o anticipados que utilizan la pila de programa |
| Tareas | Hilos independientes de ejecución que pueden llamar al procesador |
| Funciones Periódicas | Tareas ejecutadas cada cierto tiempo |
| Semáforos | Intercambio de datos sincronizados entre tareas |
| Colas | Semáforos contables |
| Relojes | Enlace de listas atómicas |
| "Streaming" | Intercambio de datos de entrada y salida para las tareas |
| Flujo de datos | Flujo de datos de entrada y salida para interrupciones de software |
| Manejo de Memoria | Baja sobrecarga en la asignación de memoria dinámica |

Los plugins de *DSP/BIOS* permiten monitorizar el programa desde el lado del anfitrión con el programa *Code Composer Studio*, utilizando variables como el uso de *CPU*, tiempos, ejecución de hilos, y más. La implementación de este paquete permite el seguimiento y la depuración a nivel de hardware (memoria de procesos, datos, registros y otros componentes pertenecientes al *DSP*), de los programas creados que se ejecuten en este procesador, lo que permite una mayor interacción con la arquitectura del *DSP* [33].

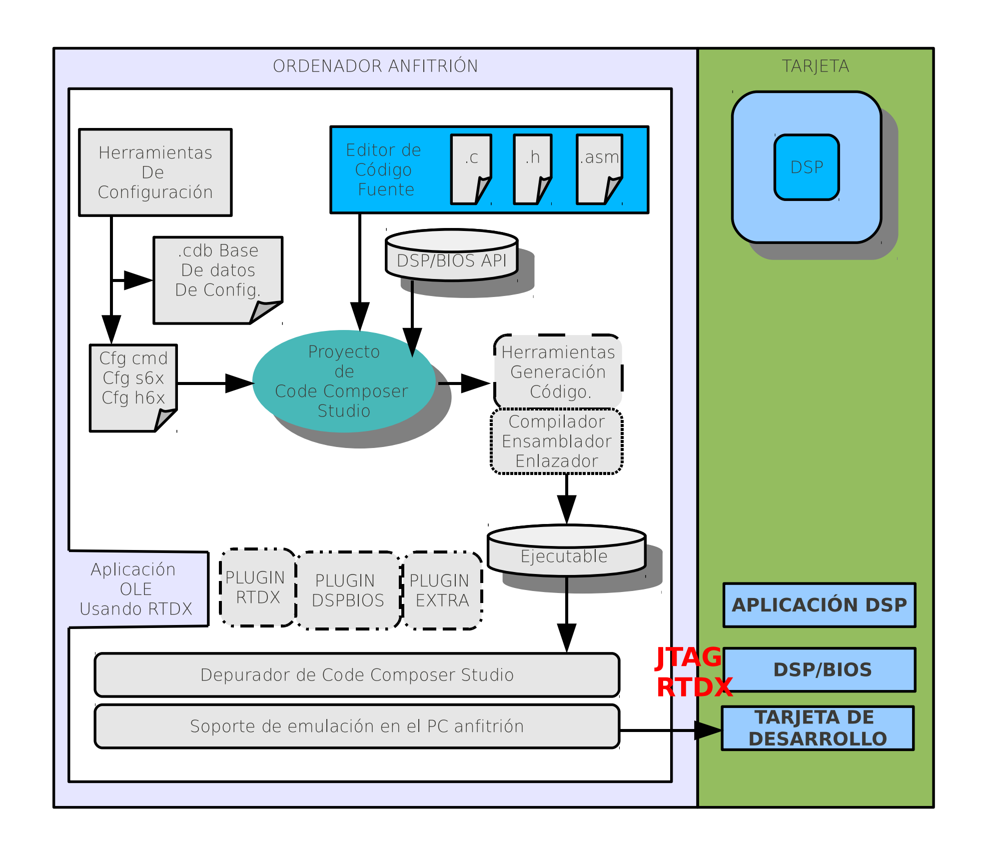



Figura 2.10.: DSP/BIOS configurado a través de CodeComposer (Tomado de [33])

El enfoque del proyecto como se introdujo anteriormente, es una combinación de procesos entre el procesador *ARM* y el *DSP*, para la ejecución de algoritmos de DSP. *DSP/BIOS* proporciona un marco de trabajo denominado *DSP/BIOS Link* que facilita la comunicación entre el procesador *ARM* de propósito general *GPP* y el *DSP* que permite el intercambio datos e información de control entre el GPP y el DSP.

#### 2.3.1.1. DSP/BIOS Link

*DSPLink* como se le llama también, está diseñado para facilitar la comunicación entre microprocesadores de propósito general y los *DSP*, a través de una capa de comunicaciones entre procesadores (*IPC*).

En el pasado, el desarrollo de aplicaciones que involucraban o residían tanto en el *GPP* como en el DSP de las arquitecturas de TI, requerían del desarrollo de software para la comunicación entre procesadores, que permitiera el intercambio de datos y de información. Además, el microprocesador debía también inicializar el *DSP* y controlar cuál algoritmo de *DSP* debía de ejecutarse para una tarea en específico. El paquete *DSPLink* permite manejar las tareas anteriomente mencionadas a través de un conjunto de servicios dedicados. Véase figura 2.11.

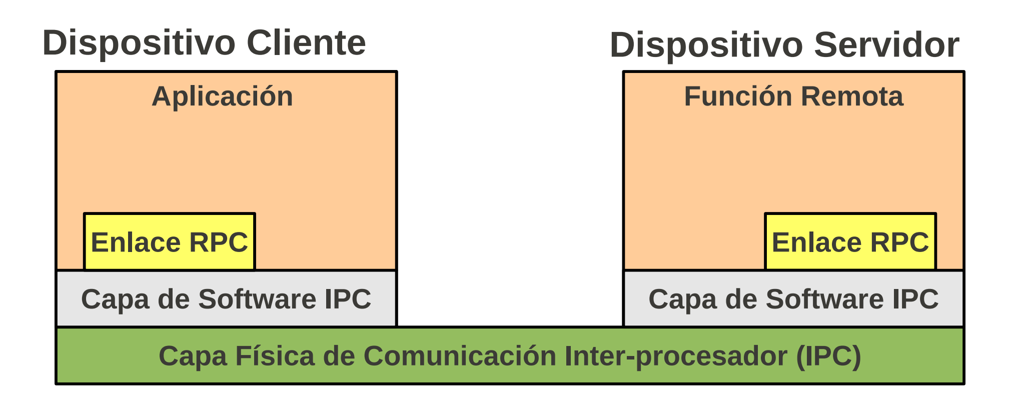



Figura 2.11.: Capas para la utilización de los protocolos IPC (Tomado de [46])

*DSPLink* está basado en protocolos de llamadas a procedimientos remotos *(RPC)*. El procesador emisor de comandos (GPP) es conocido como el dispositivo cliente y el procesador ejecutor de comandos (DSP) es conocido como el dispositivo servidor. El cliente envía comandos y parámetros al servidor sobre el medio físico de comunicaciones IPC. Una vez que el servidor completa la ejecución del comando, envía un mensaje de regreso sobre el mismo medio al cliente, que provee algún valor de retorno del procedimiento que ejecuta.

Gracias al uso de memoria compartida y señales de interrupción entre dispositivos, ambos procesadores acuerdan una dirección de memoria predeterminada para mensajes desde el ARM hasta el DSP, y viceversa. Un procesador envía mensajes al otro procesador, escribiendo el mensaje en la dirección predeterminada de memoria y envía una interrupción para señalar al otro procesador que un mensaje nuevo está disponible. El otro procesador una vez que lee el mensaje, marca una bandera en la memoria compartida, para indicar que otro mensaje puede ser puesto en esta memoria. Dependiendo de la plataforma soportada, el sistema operativo y la versión de *DSPLink*, se provee los servicios listados en la tabla 2.3.

Tabla 2.3.: Servicios que ofrece el paquete DSPLink

| Funciones | Descripción |
|---|---|
| **Control Básico de Procesadores** | Configuración del sistema para permitir el acceso a los recursos del DSP a través del GPP. Cargas del DSP con un ejecutable de DSP presente en el sistema de archivos desde el GPP. Iniciar el DSP a través del GPP. Detener el DSP a través del GPP. Carga y finalización del controlador de DSPLink. |
| **Protocolos de comunicación entre procesadores** | Protocolos completos para el manejo de diferente tipos de transferencia de datos.MSGQ: Mensajes de colas. Memoria circular cíclica. |
| **Bloques de construcción de comunicaciones entre procesadores** | Bloques de bajo nivel usado por protocolos. Cada bloque es enmascarado como una API para permitir a los desarrolladores definir sus propios protocolos en sus aplicaciones. POOL: Manejador de memoria, compartida/no-compartida.PROC_read/PROC_write: Lectura o escritura a la memoria del DSP |

*DSPLink* provee una serie de características y capacidades que lo hacen conveniente para el desarrollo de sistemas multi-núcleo, ya que encapsula detalles específicos de *hardware* de la plataforma a utilizar. Además el controlador (*dsplink.ko*) de *DSPLink* se puede añadir o integrar al kernel de *GNU/Linux* como un módulo más (véase figura 2.12) encargado de enmascarar las funciones mencionadas anteriormente, las cuales dejan al usuario exento de estas preocupaciones.

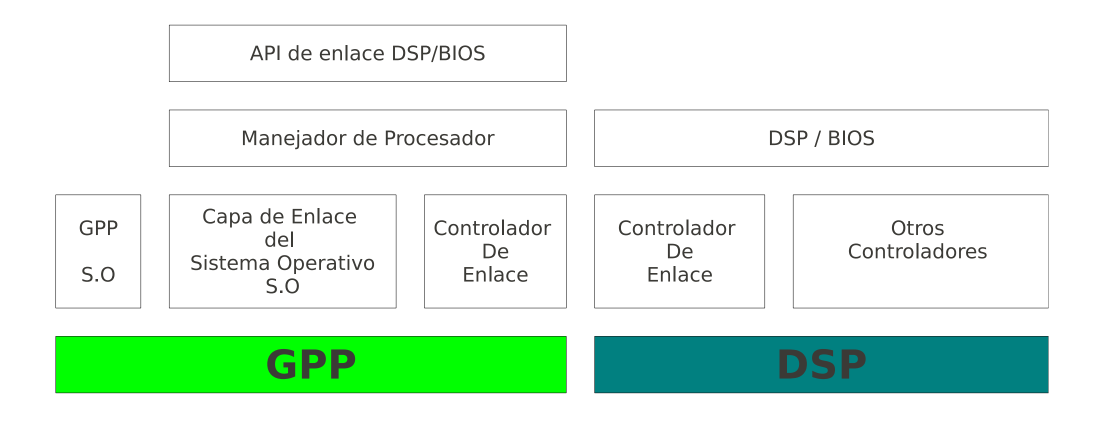



Figura 2.12.: Diagrama de la arquitectura de DSP/BIOS Link (Tomado de [43])

### 2.3.2. CMEM

*CMEM* es una *API* o biblioteca que permite el manejo de uno o varios bloques de memoria física contigua. Provee servicios para la conversión de memoria virtual a memoria física, así como una *API* para el manejo de memoria caché.

Esta memoria contigua es usada en dispositivos que no tienen *MMU* y no pueden leer direcciones virtuales, como por ejemplo el *DSP* o aceleradores de hardware *(DMA)*, los cuales comparten datos en memoria en tiempo de ejecución de algún programa. En este caso en específico se utiliza para manejar el flujo de datos entre el *DSP* y el procesador *ARM*. *DSPLink* es uno de los paquetes que utiliza memoria para poder cumplir con el paso de mensajes entre procesadores que maneja el protocolo *RPC* sobre el que está basado.

A través del uso de la configuración de bloques de memoria, *CMEM* evita a los usuarios la fragmentación de memoria, asegurando bloques físicos de memoria contigua disponibles aún después de que el sistema ha sido ejecutado por largos periodos de tiempo.

CMEM se establece como un módulo en el *kernel de GNU/Linux* a través de la instalación del controlador respectivo (*cmemk.ko*). Éste reserva un bloque de memoria *RAM* de tal modo que no se traslape con ningun otro bloque. Su construcción y configuración debe ser establecida para cada arquitectura en particular.

### 2.3.3. Herramientas XDC

Las herramientas *eXpress DSP (XDC)* contienen los implementos necesarios para crear, probar, hacer uso, e instalar componentes RTSC (Real-Time Software Components). Éstos conforman un programa diseñado para llevar el desarrollo para programadores de C, basado en componentes llamados "paquetes", y que están optimizados para sistemas embebidos de tiempo real. Además, utilizan interfaces que no son dependientes del hardware, y permiten el soporte automatizado vía scripts para la configuración de estos paquetes.

Un paquete *XDC* es una colección de archivos que forman una unidad de control de versiones. Cada paquete se establece en un directorio con un nombre en específico (y su contenido) dentro de un sistema de archivos [35].

Los usuarios de *XDC* se dividen en desarrolladores denominados: los "consumidores" y los "productores". Se puede observar en la figura 2.13, cómo se relacionan los desarrolladores de paquetes XDC.

- Los consumidores integran dentro de sus aplicaciones los denominados paquetes con contenido destinado al desarrollo de módulos para DSP, controladores de dispositivos.
- Los productores son los encargados de crear los paquetes que los consumidores utilizan.

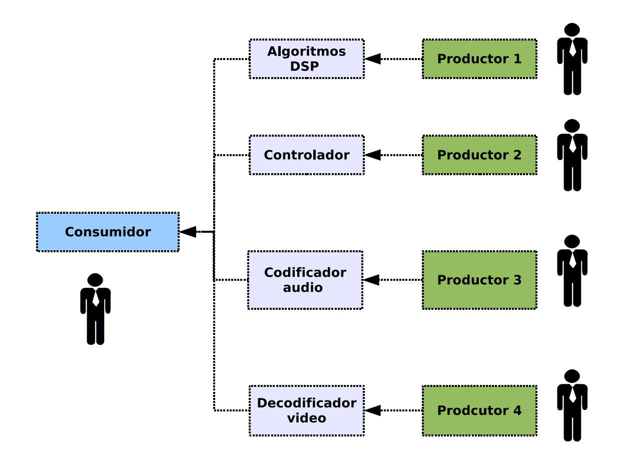



Figura 2.13.: Relación Consumidor - Productor (Tomado de [35])

*XDC* introduce herramientas y la infraestructura en un proceso de cinco fases que abarca el ciclo de vida de un "paquete" implementado en C, desde la producción hasta su consumo. En la figura 2.14 se ilustra este ciclo, y a continuación se explican cada una de las fases.

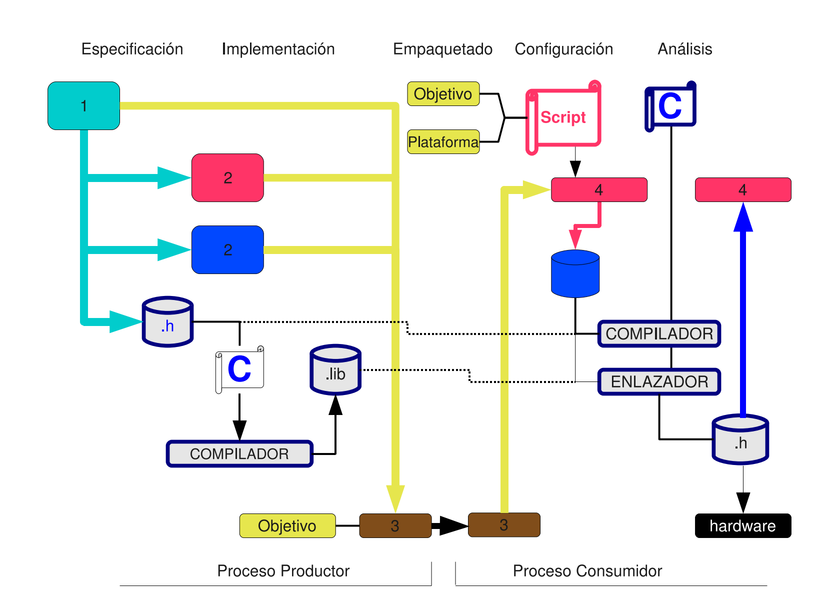



Figura 2.14.: Ciclo de un paquete XDC

1. Especificación: RTSC introduce un lenguaje de especificación especial para C que se usa para definir formalmente los límites de programación entre productor y consumidor, más alla de archivos encabezados (.h) propios del lenguaje C. El lenguaje de especificación de programación RTSC promueve tres componentes no compatibles

directamente con C, pero necesarios si se utiliza C para implementar los componentes que participan en una aplicación:

- ○ Módulo: un conjunto coherente de constantes, tipos y funciones con una especificación pública y una implementación privada.
- ○ Interfaz: un módulo abstracto (especificaciones solamente) que otras interfaces pueden heredar y que los módulos en última instancia implementan.
- ○ Paquete: un espacio de nombres de programación de alto nivel que a su vez contiene módulos e interfaces dentro de su ámbito de aplicación.

2. Implementación: Los módulos XDC están formados por dos implementaciones que se complementan:

    - ○ Implementación con la arquitectura objetivo: contenido basado en C, para un posterior uso en aplicaciones enfocadas a una plataforma de hardware específico.
    - ○ Implementación con la metainformación: contenido de alto nivel para la configuración y análisis con estas aplicaciones. RTSC introduce un meta-lenguaje basado en estándares de la tecnología JavaScript.

3. Empaquetado: los paquetes RTSC sirven como contenedores lógico y físicos. Además de proporcionar un espacio de nombres de programación que incluyen los módulos e interfaces RTSC, cada uno se integra como un directorio dentro de un sistema de archivos que siguen una serie de convenciones, pero que puede albergar archivos como bibliotecas, encabezados, scripts, documentos y hasta otros paquetes. Para ayudar a manejar el desafío del productor, el envasado RTSC simplifica el proceso de construcción de varias bibliotecas utilizando compiladores múltiples C, aprovechando las instancias de una receta que prescribe las medidas necesarias para compilar archivos de código fuente C para un conjunto particular de instrucciones y modelos de memoria de alguna cadena de herramientas de compiladores. Otros mecanismos permiten a los productores generar versiones de un paquete en particular.

4. Configuración: Desde la perspectiva del consumidor, la configuración RTSC es útil para la integración de los paquetes RTSC para el uso final dentro de un programa de aplicación que se ejecuta en alguna plataforma de hardware. El proceso comienza con una parte dedicada a comandos de configuración con el meta-lenguaje RTSC que identifica cada elemento de software requeridos para una correspondiente aplicación. Por medio de los scripts basados en JavaScript se detalla en recetas los módulos o interfaces de un paquete de RTSC que se quieren utilizar. Además se describe en estos archivos la configuración de la plataforma de hardware a la que se quiere integrar la aplicación, para compilarla y enlazarla.

5. Análisis: Además de orientar la integración de módulos RTSC en programas

ejecutables, también se puede participar activamente en el análisis en tiempo de ejecución del comportamiento del módulo, a través de un depurador en un sistema anfitrión. Por lo tanto, la creación de aplicaciones que utilizan paquetes de software basado en XDC sigue un desarrollo que se basa en las técnicas tradicionales de programación C. Los paquetes XDC estandarizan la entrega de software TI y de socios, y facilitan la integración de dicho software. Una vez que el proceso de configuración se integra en su proceso de generación, los pasos para crear una aplicación que utiliza contenido basado en XDC son los siguientes:

*a*) Configuración de la aplicación.

*b*) La escritura de código en C.

*c*) Compilar y vincular la aplicación. El objetivo principal se centra en la escritura de código C.

### 2.3.4. XDAIS (eXpressDSP Algorithm Standard)

La implementación de algoritmos de DSP depende de las políticas de manejo de memoria o mecanismos de entrada y salida específicos de cada sistema. La falta de consistencia en la integración de sistemas o de estándares de programación complica el proceso de crear un algoritmo de DSP para ser usado en más de un sistema o una aplicación sin tener que pasar por un proceso de reingeniería, reintegración y prueba.

Como parte de la solución a este problema se encuentra lo que Texas Instuments denomina como *eXpressDSP Algorithm Standard* (*XDAIS*) [34], como una forma de estandarizar los algoritmos especialmente para sus tecnologías.

Para que un algoritmo sea compatible con *XDAIS*, debe ser programado siguiendo la interfaz *IALG* [42], que se incluye dentro de este estándar y que puede ser programado en lenguaje C o ensamblador. Además, este código debe integrar una serie de reglas [41], como por ejemplo: una convención de nombres, nunca acceder directamente a dispositivos periféricos y ser código totalmente reubicable. Por medio de esta interfaz se configura el manejo de memoria del algoritmo.

También se debe seguir una interfaz específica para cada algoritmo denominada IMOD, la cual es definida por el desarrollador del algoritmo que extiende la interfaz *IALG* para poder implementar las funciones que describen la funcionalidad del algoritmo o acceder la biblioteca que implementa el algoritmo [32]. Resumiendo, un algoritmo se puede hacer compatible utilizando los siguientes pasos:

- Implementar la interfaz IALG en el código [42] con sus funciones de inicio, destrucción, reserva de memoria, control, etc.
- Implementar la interfaz específica (IMOD) en el código, aportada por el desarrollador.
- Definir las funciones principales de la estructura XDAIS, en una sección de código que se denomina V-Table, para que el compilador reconozca tales funciones.
- Verificar que se cumplan con las reglas estándares de XDAIS.
- Crear la biblioteca y el archivo de encabezado

Otros pasos adicionales son:

- Dividir la memoria en varios bloques.
- Implementar las funciones complementarias de la interfaz IALG (como por ejemplo de activación y desactivación el algoritmo).

De esta forma la funcionalidad de XDAIS se puede definir como un enmascaramiento (módulo) de las funciones del algoritmo de DSP para que una aplicación pueda administrar una instancia del algoritmo al llamar a la tabla de punteros a funciones (V-Table), esto para poder crear, eliminar y ejecutar el algoritmo. Ver figura 2.15 para una ilustración.

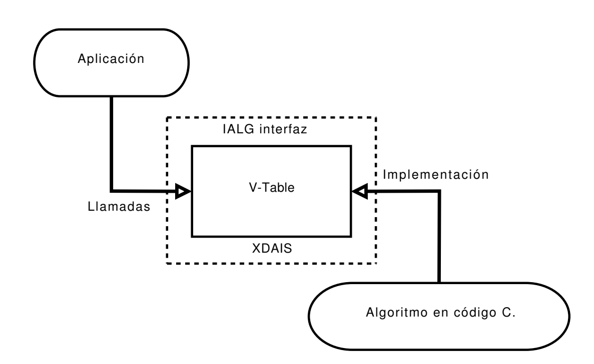



Figura 2.15.: Interfaz IALG para el estándar XDAIS

### 2.3.5. XDAIS-DM (xDM) y *iUniversal*

El estándar de *xDM* es una serie de API relacionados con diferentes módulos codecs multimedia para una fácil integración e interoperabilidad. *xDM* es construido sobre *XDAIS* [34].

En 2009 TI introduce la API de *iUniversal* en sus paquetes de software, la cual agrega una funcionalidad que extiende la utilidad del estándar VISA y permite integrar módulos que no solo impliquen algoritmos de codecs multimedia.

En la figura 2.16 se muestra la relación entre *xDM* y *XDAIS* [34], y cómo *xDM* extiende la interfaz de *IALG* de *XDAIS* dentro de interfaces dedicadas a módulos codecs de audio y video [34], y con la inclusión de *iUniversal* a módulos de algoritmos que no impliquen codecs multimedia.

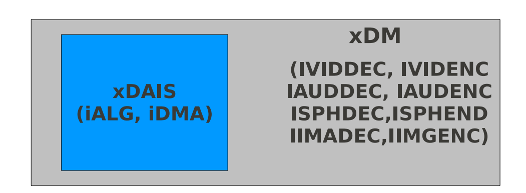



Figura 2.16.: Relación entre xDM y XDAIS (Tomado de [34])

*xDM* define nueve interfaces. Ocho de éstas son dedicadas a codecs multimedia y son conocidas con el nombre de *VISA* (Video, Image, Speech, Audio):

- IVIDENCx Interfaz genérica para codificadores de video
- IVIDDECx: Interfaz genérica para decodificadores de video
- IAUDENCx: Interfaz genérica para codificadores de audio
- IAUDDECx: Interfaz genérica para decodificadores de audio
- ISPHENCx: Interfaz genérica para codificadores de voz
- ISPHDECx: Interfaz genérica para decodificadores de voz
- IIMGENCx: Interfaz genérica para codificadores de imágenes
- IIMGDECx: Interfaz genérica para decodificadores de imágenes

La novena interfaz definida en estos estándares es:

- iUNIVERSALx: interfaz genérica para cualquier tipo de algoritmo

La interfaz *XDAIS* permite tener una comunicación entre una aplicación y un algoritmo de DSP como lo muestra la figura 2.17, con su interfaz *IMOD* y *IALG*. Cuando se incluye el estándar xDM dentro de *XDAIS*, el algoritmo utiliza una de las nueve interfaces estándar predefinidas en este paquete. *xDM* es un superconjunto de la interfaz *IALG*.

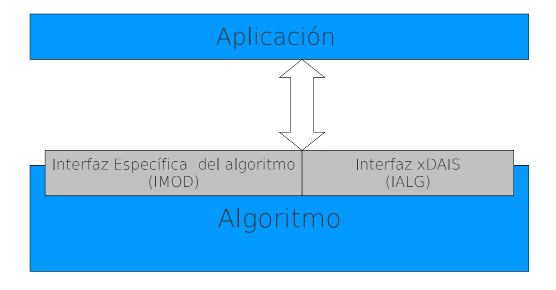



Figura 2.17.: Interfaz de *XDAIS* (Tomado de [34])

A través de éstas, se puede ajustar un determinado algoritmo o aplicación mediante la extensión de la interfaz *IMOD* a la interfaz xDM. En casos más sencillos, *IMOD* será idéntico a la interfaz *xDM*. Lo anterior se puede observar en la figura 2.18.

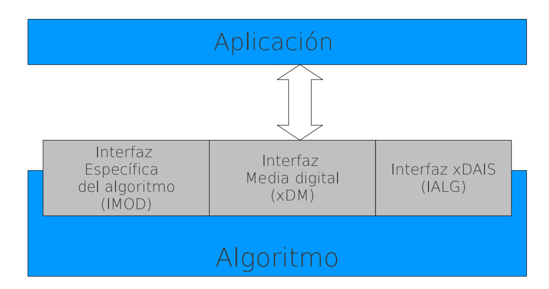



Figura 2.18.: Interfaz de *xDM* para un algoritmo (Tomado de [34])

### 2.3.6. Codec Engine

*Codec Engine* [36] es un marco de trabajo que permite a las aplicaciones instanciar y trabajar con algoritmos usando una *API* común.

Esta API posee las siguientes características:

- Algoritmos que se ejecutan localmente (en el GPP) o remotamente (en el DSP)
- Plataformas que pueden ser conformadas por un GPP+DSP, un DSP solamente, o un GPP
- Todos los dispositivos GPP y DSP soportados utilizan la misma API
- Todos los sistemas operativos soportados utilizan la misma API

Desde la perspectiva del usuario, basada en la construcción de una aplicación, *Codec Engine* es un conjunto de bibliotecas que se utilizan para instanciar módulos de algoritmos con el estándar de *XDAIS*, por ende, también es capaz de instanciar módulos de algoritmos

*xDM*[34]. Esto a través de una *API* para el manejo de los estándares *VISA y iUniversal* [36].

*Codec Engine* es un paquete diseñado para resolver incógnitas asociadas al desarrollo de aplicaciones en un Sistema en Chip dado, dentro de las cuales destacan:

- El manejo de memoria para DSP
- Comunicación inter procesadores
- El control de tareas que se ejecutan en tiempo real.
- La depuración en procesadores heterogéneos.
- Eficiencia en el desarrollo de algoritmos

Desde el punto de vista del desarrollador *Codec Engine* especifica cuál algoritmo debe ejecutarse y en que momento; agrega nuevos algoritmos, usando los correctos estándares y técnicas y provee una API independiente de la plataforma o del módulo algoritmo que se quiera crear.

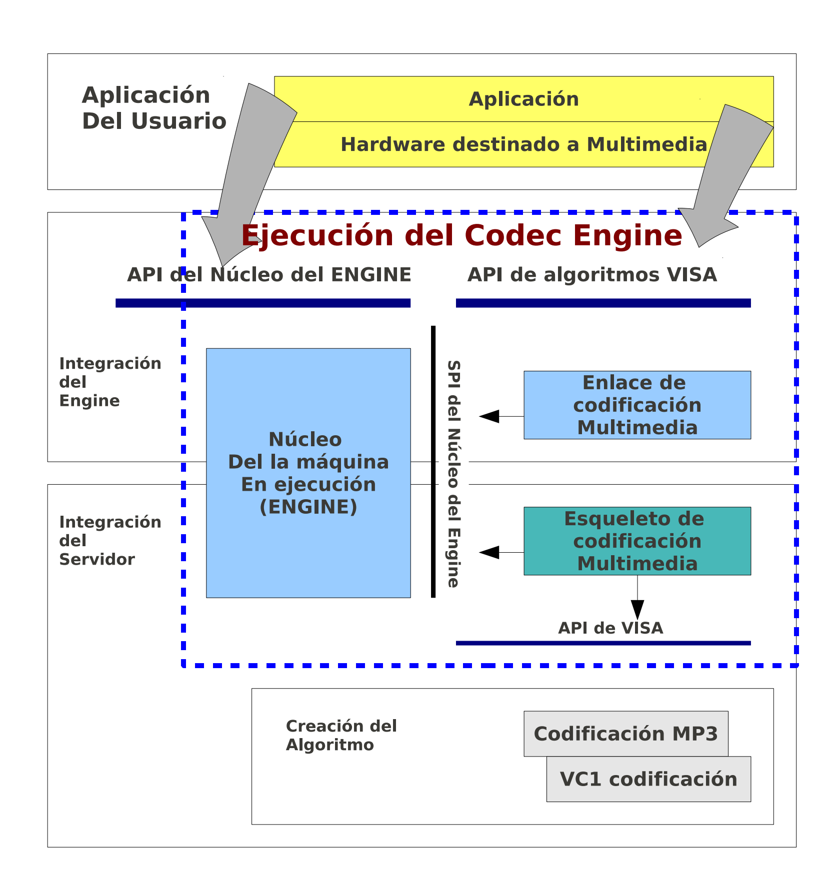
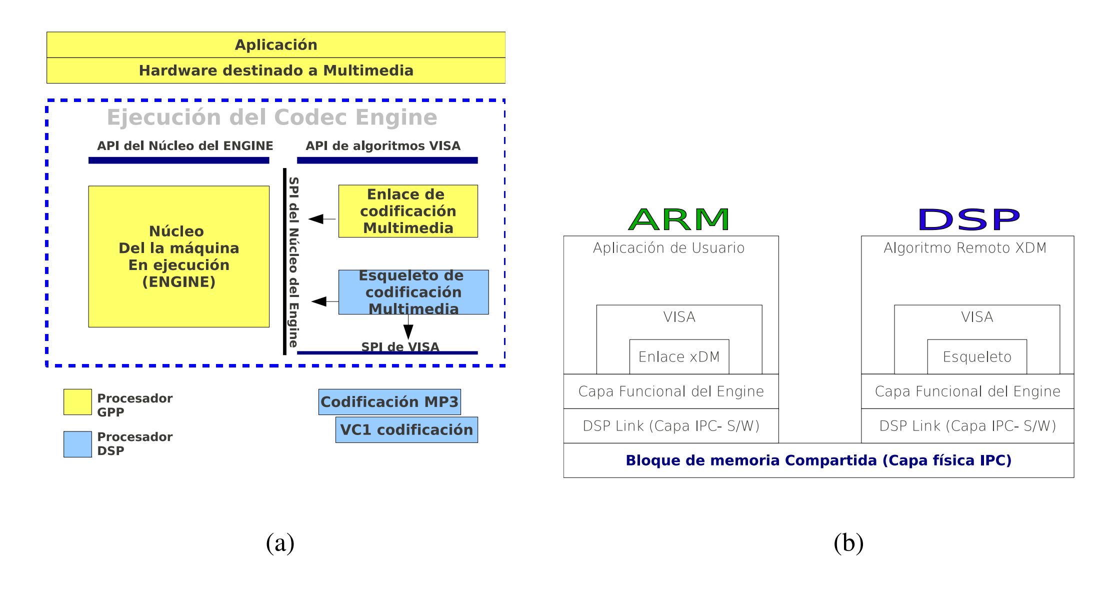



Figura 2.19.: Arquitectura del marco de trabajo de CodecEngine (Tomado de [36])

La figura 2.19 muestra la arquitectura que presenta una aplicación al incluir las *API* de *CodecEngine* para la ejecución de un algoritmo.

El marco de trabajo de *Codec Engine* utiliza el concepto de *RPC* (pues utiliza la capa de *DSPLink*) con capas de software que son una API encargada de enmascarar funciones de los estándares *VISA* y de *iUniversal*. Dicho marco de trabajo se integra en conjunto con la capa funcional denominada motor central de ejecución (*Core Engine Runtime*) o capa funcional de la máquina. De esta forma el motor central de ejecución se encarga de manejar instancias a objetos[1] de algoritmos encapsulados como módulos xDM. Estas instancias pueden ser hechas remotamente o localmente, dependiendo de la arquitectura que se utilice, ya sea ARM, o ARM-DSP.

Las capas de enlace de *VISA* o *iUniversal* definen el procedimiento para crear, borrar o usar un módulo remoto o local (ejecutado en el DSP o en el GPP respectivamente), siempre y cuando este haya sido creado con los estándares *xDM* [46]. Por lo tanto, *Codec Engine* funciona como un conducto para transferir las llamadas hechas por parte de la aplicación con los enlaces de *VISA* o *iUniversal* a las funciones remotas del algoritmo también creados con los esqueletos estándares de *VISA* o *iUniversal*.




(a) (b)

Figura 2.20.: Marco de trabajo de CodecEngine en una arquitectura dual (Tomado de [36])

El funcionamiento de este marco de trabajo en una arquitectura con *GPP* y *DSP*, se puede observar en la figura 2.20. Aquí, se muestran los procesos que se ejecutan en el procesador GPP y los que se ejecutan en el DSP. Los enlaces y los esqueletos son los encargados de realizar el proceso de comunicación entre la aplicación (ejecutada desde el GPP) y el

[1] Con objeto se refiere al concepto tal y como se hace en lenguajes orientados a objetos, como C++, Java.

algoritmo de DSP, a través de la capa de *DSPLink*. Por lo tanto se tiene un procesador cliente (ARM) y un procesador servidor (DSP).

En este caso, el motor central de ejecución se divide en una sección que se encuentra en el procesador ARM y la otra en el DSP. La parte que se encuentra del lado del DSP se denomina comúnmente *CodecServer* (funciona como un servidor remoto, que comunica los dos frentes) y será explicada en la siguiente sección.

#### 2.3.6.1. Codec Server

Este paquete se utiliza solamente cuando se tienen arquitecturas duales (DSP+GPP) y se desea que algún algoritmo sea ejecutado en el DSP (remotamente). El *Codec Server* es un archivo ejecutable que integra los algoritmos creados con el estándar *XDAIS* y sus marcos de trabajo (por ejemplo *DSP/BIOS*, y los controladores de *DSPLink*) que se carga y se inicia en memoria RAM y es ejecutado por el DSP.

El servidor puede ser utilizado por las aplicaciones que se encuentran en el GPP para acceder información acerca del servidor del DSP y para controlarlo. Más aún, estas API permiten a una aplicación de GPP obtener información sobre el número de bloques de memoria configurada en el servidor de DSP, conocer el uso actual de un bloque individual de memoria, y reconfigurar la base y el tamaño del servidor de DSP para los algoritmos [36].

La creación del *Codec Server* debe tener dos pasos de configuración: uno que permita configurar *DSP/BIOS* a través de un archivo llamado TConf script, y el otro que configure el resto de los paquetes. Esto a través de las herramientas de compilación *XDC* sobre los componentes de *CodecEngine*, *DSPLink* y los algoritmos módulos basados en XDAIS.

El *Codec Server* funciona como una interfaz de configuración que se encarga de crear un servidor, utilizado para añadir los algoritmos dentro del DSP, y además manejar las respectivas referencias de los algoritmos en tiempo de ejecución.

### 2.3.7. DMAI

DMAI es una capa de software que se encuentra encima del sistema operativo (*Linux o DSP/BIOS*) y de *Codec Engine* como una forma de asistir aplicaciones portables dentro del marco de *Codec Engine* y la tecnología *DaVinci*.

Esta capa se basa en módulos que usando una API con una abstracción de búfer, no solo enmascara datos de video, audio, voz, sino que también poseen información que describen al búfer (tamaño, tipo, direcciones, características), utilizado por *CodecEngine* para poder realizar el manejo de datos entre el *GPP* y el *DSP*. En la figura 2.21 se muestra el nivel

que ocupa la capa de software de *DMAI*. Puede observarse que la aplicación puede escoger cuándo usar *DMAI*, el sistema operativo o Codec Engine:

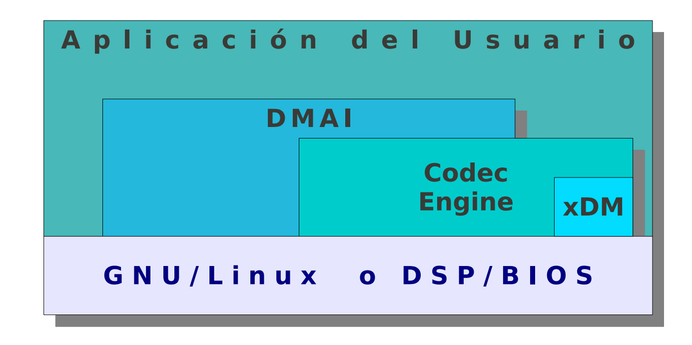



Figura 2.21.: Bloque del Diagrama de *DMAI*

### 2.3.8. Herramientas de Generación de Código (CodeGen Tools)

El desarrollo de la industria DSP ha llevado a TI a crear un módelo de validación para compilar código C en una estación de trabajo y luego portar los ejecutables a una tarjeta de desarollo compatible . Este modelo es muy similar al compilador de la cadena de herramientas analizada en la sección 2.2.4. Sin embargo las herramientas de generación de código, no solo compilan código para el DSP sino que se encargan de optimizarlo basándose en banderas de compilación.

Texas Instruments recomienda la versión CGTc6000 como compilador cruzado para el desarrollo de software para DSP. Este paquete ofrece ventajas significativas al usuario, ya que deja al compilador realizar la selección de la instrucción más adecuada. Además controla el paralelismo, la canalización del flujo de datos y el direccionamiento de registros (*parallelizing, pipelining, and register allocation*) [44]. Esto permite que el usuario se centre en su aplicación y no en el manejo de capas inferiores. Gracias a este control, el código se vuelve mantenible y modificable, ya que todo está basado en un estándar. Las CGTools se describen en el documento "*TMS320C6000 Optimizing C Compiler* "[44].

## 2.4. Concepto de Video Digital

El video digital es parte integral en aspectos de negocios, educación y entretenimiento, desde la televisión digital hasta los videos a través de la web. El video digital es información visual representada en forma discreta, adecuada para el almacemamiento y/o la transmisión a través de dispositivos electrónicos digitales[10]. A continuación se describe una serie de conceptos relacionados al despliegue de video digital.

### 2.4.1. Dominios Espacio-Temporal

Una imagen es una proyección de una escena tridimensional en un plano 2-D, para un instante particular de tiempo, como se muestra en la figura 2.22. La secuencia de video representa la escena sobre un periodo de tiempo determinado utilizando imágenes como muestras en el tiempo del espacio.

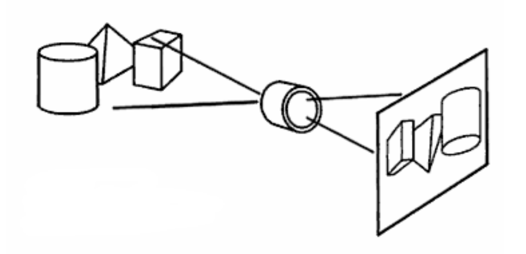


Figura 2.22.: Imagen de una escena 3-D en una imagen de video

Una imagen digital es generada a través del muestreo de una señal espacial analógica. La representación más común para una imagen muestreada es un rectángulo o cuadrícula, que posee un largo y un ancho, con los puntos muestreados posicionados en esta cuadrícula. A esto se le denomina matriz de pixeles, como se muestra en la figura 2.23.

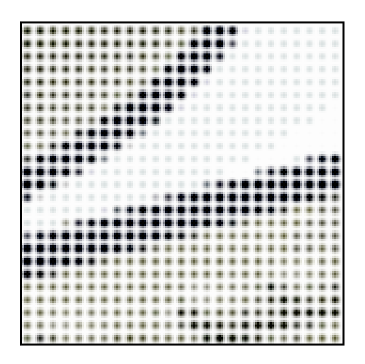


Figura 2.23.: Muestreo Espacial

Cada pixel es una muestra espacial, con un valor determinado, que representa el brillo (luminancia) y el color de una muestra. Entre mayor sea la cantidad de puntos de muestreo mayor debe ser la capacidad de almacenamiento de las muestras (memoria) y menor el tiempo para tratar cada muestra en aplicaciones de video.

Una escena real es continua tanto en tiempo como en espacio. Para obtener una señal de video digital se necesita muestrear la escena real tanto en el espacio proyectado como en tiempo

(figura 2.24). Una secuencia de imágenes (muestreo temporal) captadas y desplegadas a una tasa de por lo menos 24 a 60 imágenes por segundo permite crear una ilusión de movimiento fluido. En la tablas 2.5 y 2.6 se muestran tamaños de imágenes y tasas de muestreo temporal para video digital respectivamente.

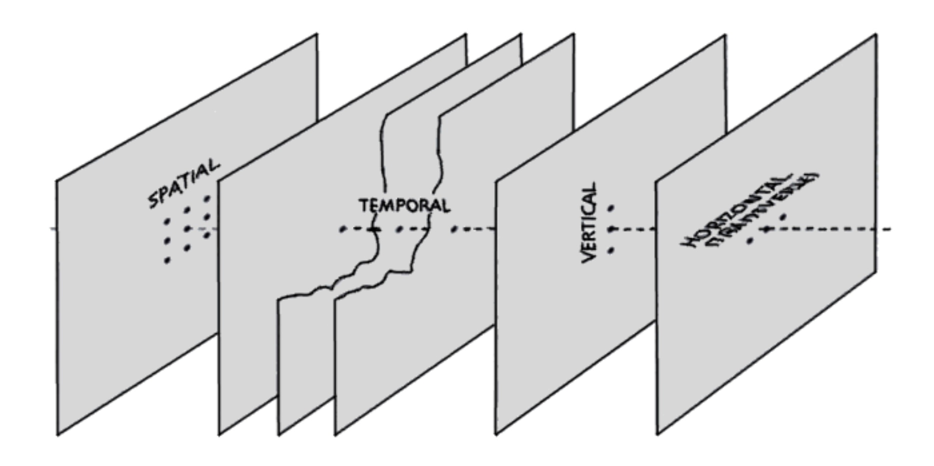



Figura 2.24.: Dominios espacio-temporal de video digital. (Tomado de [10])

Tabla 2.5.: Tamaños de imágenes en video. Tomado de [10]

| **Tamaño de la imagen** | **Número de puntos de muestreo** | **Equivalente en video análogo** |
|---|---|---|
| $352 \times 288$ | 101376 | Video VHS |
| $704 \times 576$ | 405504 | Transmisiones de televisión en formatos como PAL o NTSC |
| $1440 \times 1152$ | 1313280 | Televisón Alta Definición |

Tabla 2.6.: Tasas de muestreo temporal de un video digital. Tomado de [10]

| **Tasa de video (cuadros/s)** | **Apariencia** |
|---|---|
| 10 | Lento, movimiento poco natural |
| 10-20 | Movimientos lentos se ven bien, no así los rápidos |
| 20-30 | Movimiento razonablemente fluido |
| 50-60 | Movimineto muy fluido |

### 2.4.2. Despliegue de video y Espacios de Colores

A pesar de que se puede describir el color mediante la medición de su distribución espectral de potencia (la intensidad de la radiación electromagnética visible) para la percepción humana esto conlleva cierto grado de redundancia, puesto que las muestras de color que toma la retina del ojo utiliza sólo tres bandas, que corresponde aproximadamente a la luz roja, verde y azul. La sensibilad a señales de color (conos), junto con los de la intensidad, se combinan en el cerebro para dar varias "sensaciones" del color [2].

Entre las más importantes sensaciones se pueden mencionar:

- Luminosidad: percepción de un área con más o menos luz
- Matiz: asociado a la frecuencia dominante del espectro percibido
- Croma o saturación: relacionado con la pureza de la frecuencia dominante

En el despliegue de imágenes o videos digitales se utiliza el concepto de espacios de color, que son modelos matemáticos abstractos mediante los que se identifica el color a través de tuplas de números, por lo general 3 ó 4; por ejemplo el color azul se puede representar por la tupla (0,0,255) en el espacio RGB, o por (240,100,100) en el espacio HSI.

Una imagen monocromática puede ser representada usando solamente una dimensión de color. Esta dimensión indica el brillo o la luminancia de cada muestra, que lo que describe es escala de grises. Una muestra con solo esta dimensión es representada por $n$ bits siendo 0 el valor de negro y $2^n - 1$ el valor del blanco. Con solo tener la componente de luminancia en una imagen es posible que el sistema visual humano reconozca las figuras presentes, ya que éste es más sensible a la luminancia que al color. La luminancia es generalmente representada en 8 bits, pero para aplicaciones especiales (Rayos X) se pueden utilizar 12 bits.

A continuación se explica los espacios de color RGB y $YC_rC_b$ que conciernen a este proyecto.

### 2.4.3. Espacio RGB

El espacio RGB es un módelo de color basado en la síntesis aditiva con el que es posible representar un color mediante la adición de los tres colores de luz primarios: rojo, verde y azul. El modelo de color RGB no define por sí mismo lo que significa exactamente rojo, verde o azul; es lo que se conoce como espacio de color no absoluto. Éste es el espacio de color producido en una pantalla CRT (tubo de rayos catódicos) cuando se aplican los valores de los píxeles en una tarjeta de gráficos. RGB puede ser visualizado como un cubo con los tres ejes correspondientes a rojo, verde y azul. Se puede visualizar ejemplos acerca de las

combinaciones RGB en la tabla 2.7. Usando este sistema son necesarios entonces $3 \times 8 = 24$ bits para representar el color de un pixel [2].

Tabla 2.7.: Tabla del color para RGB

| $Valor_{Rojo}$ | $Valor_{verde}$ | $Valor_{azul}$ | $Valor_{HEX}$ | $Color$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 000000 | Negro |
| 255 | 255 | 255 | $FFFFFF$ | Blanco |

#### 2.4.3.1. $RGB_{565}$

El espacio de colores $RGB_{565}$ es un espacio de 16 bits por pixel, formado por 5 bits para el rojo, 6 bits para el verde y 5 bits para el azul, como se muestra en la figura 2.25. Para calcular el tamaño en bytes de una imagen representada en el formato $RGB_{565}$ se utiliza

$$t_{bytes} = 2 \times w \times h \tag{2.1}$$

Donde $h$ es la altura, y $w$ es el ancho de la imagen en pixeles a desplegar.

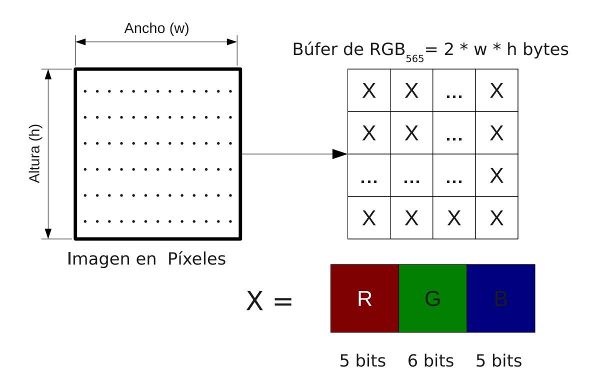



Figura 2.25.: Formato RGB565

### 2.4.4. $YC_bC_r$

La vista humana es más sensible a la luminancia que al color y es más eficiente representar el color en una imagen separando la luminancia de la información de color. RGB no puede tomar ventaja de esto, ya que en cada componente de color de ese espacio existe luminancia. Un espacio de color que toma ventaja de esto es el $Y : C_r : C_b$, donde Y es la componente

de luminancia, $C_r$ y $C_b$ son conocidos como la diferencia de color o crominancia. $C_r$ es la crominancia en rojo y $C_b$ la crominancia en azul. Relacionado con el espacio de color RGB la crominancia se comporta representando la variación de la intensidad de color y el fondo que da la luminancia de la imagen.

$$C_r = k_r * (R - Y) \tag{2.2}$$

$$C_b = k_b * (B - Y) \tag{2.3}$$

$$C_g = k_g * (G - Y) \tag{2.4}$$

La suma de $C_r$ + $C_g$ + $C_b$ es una constante por lo que solo 2 de las 3 componentes de crominancia se codifican. La tercera puede ser calculada de las otras dos. Por eso es que este espacio solo utiliza $C_r$ y $C_b$ como componentes de crominancia. En este formato la componente de luminancia se transmite con todo detalle, mientras a las componentes de crominancia se les puede reducir el detalle haciendo un submuestreo (filtrado, o promedio), reduciendo así la cantidad de datos requeridos sin tener una pérdida de calidad visual en la imagen.

#### 2.4.4.1. Formatos de submuestreo de crominancia

El submuestreo en una imagen de video para el formato $YC_bC_r$ es designado por una cadena de 3 o 4 enteros separados por dos puntos L:a:b:c, como se muestra en la figura 2.26. La relación entre los números enteros indica el grado de submuestreo vertical y horizontal.

- L: Muestreo de referencia horizontal para la componente de luminancia Y. Generalmente el valor es 4, lo que representa es que por cada pixel existe una muestra de luminancia.
- a: Número de muestras relativas horizontales de crominancia(Cr,Cb) con respecto a la luminancia. Si es 4, representa que se va a tener la misma cantidad de muestras horizontales de Cb y Cr con respecto a Y. Un 2 significa que se va a tener la mitad de muestras horizontales con respecto a Y.
- b: Si este tercer dígito es igual al segundo, no hay un submuestreo vertical. Si el valor es cero, hay un submuestreo vertical de 2:1 tanto de la muestra Cb como de Cr.
- d: Si aparece este dígito debe ser idéntico al primero, indicando la presencia de una cuarta señal que contiene información de transparencia (canal alfa), con muestreo idéntico a la luminancia.

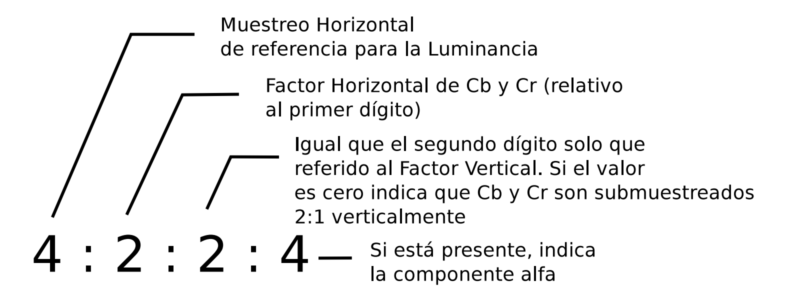



Figura 2.26.: Submuestreo de crominancia

En la figura 2.27 se observan diferentes configuraciones de los formatos anteriormente mencionados.

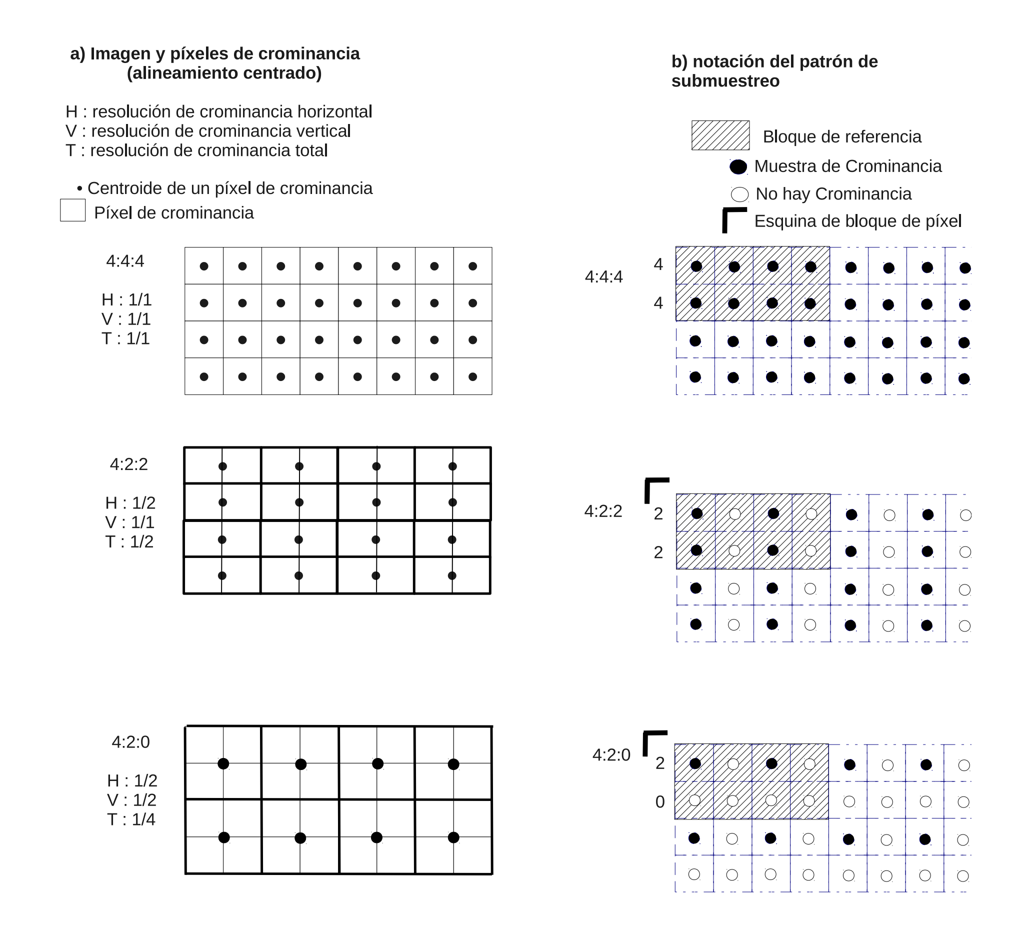



Figura 2.27.: Muestreos de formatos YCbCr. (Tomado de [5])

El submuestreo 4:4:4 significa que las tres componentes $(Y : C_r : C_b)$ tienen la misma resolución y por ende cada componente existe en cada posición de un pixel. El número indica el muestreo relativo de cada componente en la dirección horizontal, por ejemplo por cada 4 muestras de luminancia existen 4 de $C_r$ y de $C_b$.

En el submuestreo 4:2:2, el número indica que por cada 4 muestras de luminancia horizontal hay 2 muestras de $C_r$ y 2 muestras de $C_b$. El submuestreo 4:2:0 indica que $C_r$ y $C_b$ tienen la mitad de resolución tanto vertical como horizontal [10].

Con respecto al almacenamiento de los datos los formatos de $YC_bC_r$ se pueden clasificar en planares o empaquetados. El formato empaquetado tiene muestras de Y, $C_b$ y $C_r$ que se encuentran agrupadas en macropixels almacenados en una sola matriz, y los formatos planares almacenan cada componente por separado como una matriz, y la imagen final es una fusión de los tres planos distintos. Un formato empaquetado tipo 4:2:2 tiene la configuración mostrada en la figura 2.28.

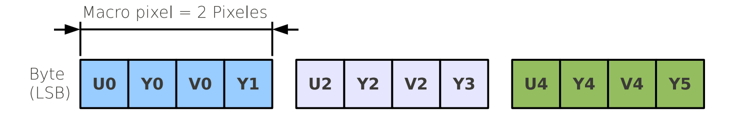



Figura 2.28.: $YC_bC_r$ empaquetado 4:2:2

El formato $I_{420}$ es un formato planar de la forma 4:2:0. En la figura 2.29 se muestra su configuración. El cuadro completo representa un bufer de $I_{420}$, y cada subsección representa muestras de $Y$, $C_b$, $C_r$ que tienen un tamaño de 1 byte cada una. Para calcular el tamaño en bytes de una imagen representada en este formato se utiliza

$$t_{bytes} = 3/2 \times w \times h \tag{2.5}$$

Donde $h$ es la altura, y $w$ el ancho de la imagen en pixeles a desplegar.

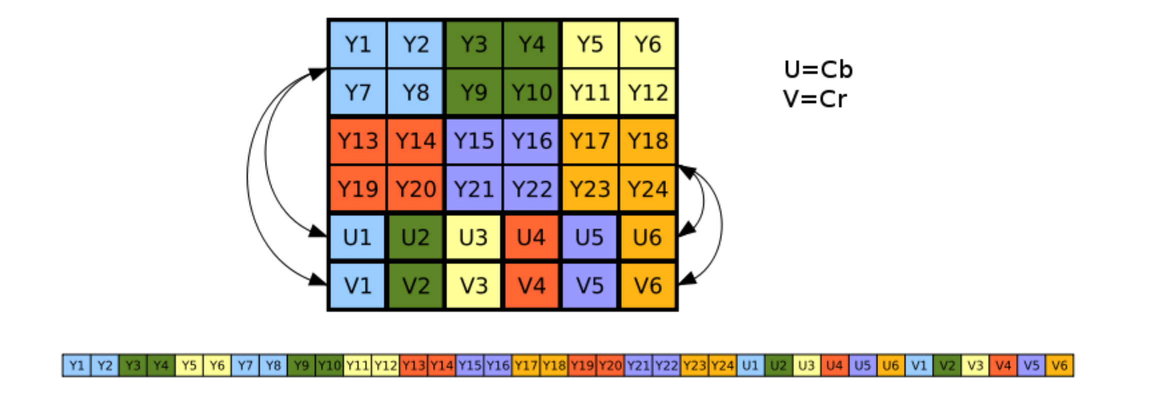



Figura 2.29.: Estructura de un búfer del formato I420

### 2.4.5. Conversión de YCbCr a RGB

La conversión entre estos espacios implica el despeje de (2.2), (2.3), (2.4) y teniendo el concepto de que la suma de las tres crominancias es una constante se llega a una forma de obtener RGB a través de $YC_bC_r$.

$$R = Y + k_1 C_r \tag{2.6}$$

$$G = Y - k_2 C_b - k_3 C_r \tag{2.7}$$

$$B = Y + k_4 C_b \tag{2.8}$$

Se puede observar que para obtener G se puede extraer de la representación $Y : C_r : C_b$ restando $C_r$ y $C_b$ a $Y$. Las constantes se definen de acuerdo al estándar que se quiera tener. Las constantes $k_1$, $k_2$, $k_3$, y $k_4$ se encuentran relacionadas con las constantes de (2.2), (2.4) y (2.3).

## 2.5. Audio Digital

El sonido es una onda mecánica longitudinal que se propaga a través de un medio elástico. Fisicamente el sonido esta conformado por oscilaciones de la presión de aire que son capturadas y percibidas en el cerebro. Para que dichas oscilaciones sean audibles por un ser humano, éstas deben estar aproximadamente entre los 20Hz y los 20kHz. El audio digital es la codificación de una señal eléctrica que representa una onda sonora y se obtiene a partir de dos procesos: el muestreo y la cuantificación digital de la señal eléctrica [49].

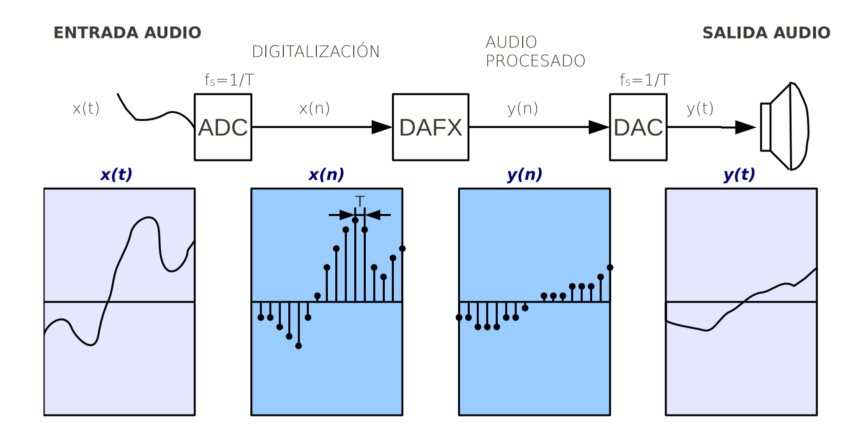



Figura 2.30.: Digitalización y procesamiento de una señal (Tomado de [49])

Actualmente existen dispositivos que permiten almacenar, procesar y reproducir sonidos. Se muestra en la figura 2.30 el proceso de digitalizacion que sufre una señal analógica, así como el proceso al cual debe someterse hasta que es reproducido. Existen dos factores que determinan la calidad de una grabación digital: la frecuencia y el formato de la muestra.

### 2.5.1. Frecuencia de muestreo

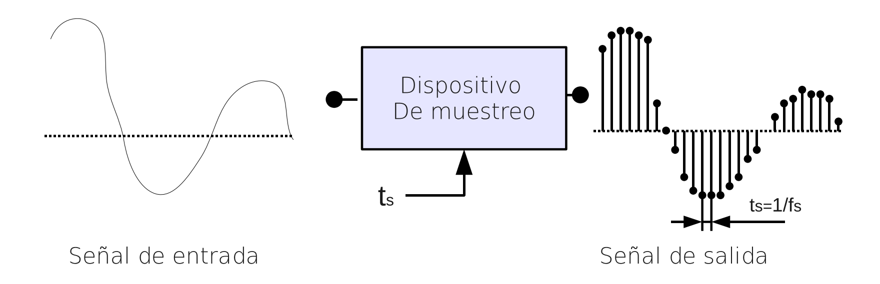



Figura 2.31.: Muestreo de una señal analógica (Tomado de [17])

La conversión analógica-digital (conversión $A/D$) consiste en realizar de forma periódica medidas de la amplitud (tensión) de una señal en un periódo de tiempo definido, en la figura 2.31 se muestra un proceso de muestreo de una señal analógica a una frecuencia de muestreo $f_s$ [17].

Según el criterio de Nyquist, cuando una señal es muestreada, la frecuencia de muestreo debe de ser al menos dos veces la el ancho de banda de la señal original [17]. En la realización de este proyecto se desean digitalizar señales de audio dentro del rango audible, es decir, de 20 Hz a 20 kHz. Bastaría con hacer uso de una frecuencia de muestreo equivalente a 40000 Hz para obtener una respuesta valida de acuerdo al teorema de Nyquist. Sin embargo en aplicaciones de audio se utiliza una frecuencia de 44100 Hz [17]. Una frecuencia de muestreo con un margen de 4100 Hz permite compensar las pérdidas por la pendiente finita entre la banda de paso y la banda supresora que tienen los filtros utilizados (anti-aliasing) durante la conversión analógica-digital.

### 2.5.2. Formato de la muestra

El formato de la muestra define el tamaño de cada muestra en bits *(width)*, así como como la precisión de la misma *(depth)*. El formato de la muestra determina la precisión en función de la amplitud de las muestras de audio. El tamaño de las muestras necesario en un proceso de cuantificación digital se determina a partir del análisis del ruido y de la intensidad máxima a registrar [49].

En la figura 2.32 se detalla el manejo de un búfer de audio y la manipulación de la muestra según el tamaño y la precisión de la misma. Por ejemplo un disco compacto de audio utiliza un tamaño por muestra de 16 bits con una precisión de 16 bits, lo que corresponde a alrededor de 5 dígitos decimales de precisión.

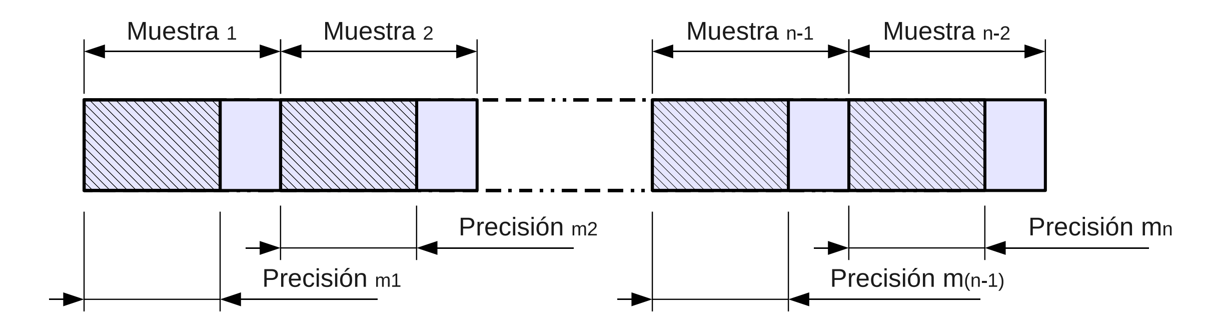



Figura 2.32.: Tamaño y precisión de la muestra de audio

### 2.5.3. Manejo del búfer de audio

El manejo de los datos se define según una serie de caracteríticas presentadas a continuación:

- Valor del tamaño de la muestra
- Valor de la precisión
- Tamaño en bytes del búfer

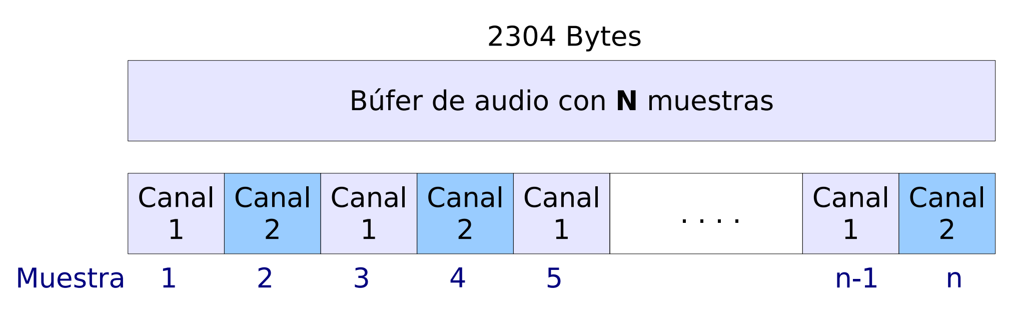



Figura 2.33.: Manejo del flujo de datos de audio

La figura 2.33 ilustra un caso concreto sobre el manejo del flujo de información de audio estandár establecido por la sociedad de ingeniería en audio (AES)[50].

Dados $B_b$ el número de bytes por búfer, $B_m$ el número de bytes por muestra, el número total de muestras por búfer $M_B$ esta dado por

$$M_B = B_b \times \frac{1}{B_m} \tag{2.9}$$

Si se condiciona al hecho que el audio utilizado es estéreo (2 canales de audio) y que cada muestra es representada con 2 bytes por muestra con precisión del 100 % del tamaño de la muestra (2 bytes) con un alineamiento entrelazado y una frecuencia de muestreo de 44100 Hz se puede describir el caso aplicando

$$M_B = (2304\,bytes) \times \frac{1}{(2\,bytes)} = 1152\,muestras$$

Por otro lado a partir de los parámetros antes mencionados junto a $f_s$ la frecuencia de muestreo, $B_m$ el número de bytes por muestra y $B_b$ el número de bytes por búfer, se puede obtener el tiempo total por búfer $t_B$

$$t_B = B_b \times \frac{1}{f_s} \times \frac{1}{B_m} \tag{2.10}$$

Aplicando (2.10)

$$t_B = (2304\,bytes) \times (\tfrac{1}{44100\,Hz}) \times (\tfrac{1}{2\,bytes}) = 0{,}02612\,s = 26{,}12\,ms$$

Sea $n_C$ el número de canales utilizados, $t_B$ el tiempo total por búfer y $M_B$ el número total de muestras por búfer, se obtiene el retardo existente entre muestras $t_{\triangle t}$ con

$$t_{\triangle t} = n_C \times \left( t_B \times \frac{1}{M_B} \right) \tag{2.11}$$

Aplicando (2.11)

$$t_{\triangle t} = 2 \times \left( 26{,}12\, ms \times \frac{1}{1152\, muestras} \right) = 45\, us$$

### 2.5.4. Filtro FIR tipo peine (comb filter)

#### 2.5.4.1. Filtro FIR

Se ha seleccionado para ejemplificar un algoritmo de audio procesado en el DSP de la OMAP-L138 la implementación de un filtro FIR tipo peine (Comb filter). Un filtro digital es un sistema que dependiendo de sus parámetros realiza un proceso de discriminación de una señal de entrada obteniendo variaciones en su salida. Comúnmente se utiliza para atenuar o amplificar frecuencias [12].

Los filtros digitales de Respuesta Impulsiva Finita o filtros FIR son filtros que a la salida producen un número finito de términos, siempre y cuando la entrada sea finita. Estos filtros tienen todos los polos en el origen, por lo que son estables. La estructura de la señal a la salida del filtro se basa en la combinación lineal de las entradas actuales y anteriores y se puede expresar de forma general como la convolución de la entrada $x(n)$ con la respuesta a impulso del filtro $h(n)$ [12]:

$$y(n) = \sum_{k=0}^{M-1} h(n) \cdot x(n-k) = h(n) * x(n) \tag{2.12}$$

donde $M$ es el orden del filtro, que también coincide con el número de coeficientes del filtro. Aplicando la transformada $z$ a la respuesta al impulso del filtro FIR $h(n)$, donde se tiene:

$$H(z) = \sum_{k=0}^{M-1} h(k) z^{-k} = h(0) + h(1)z^{-1} + \ldots + h(M-1)z^{-(M-1)} \tag{2.13}$$

Su representación en diagrama de bloques esta dada en la figura 2.34.

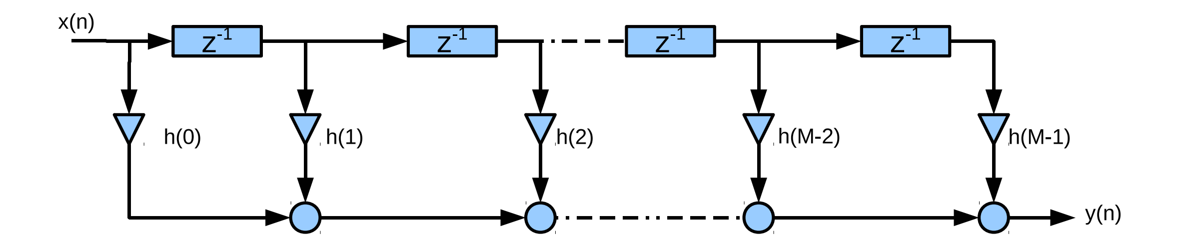



Figura 2.34.: Implementación de un filtro FIR. (Tomado de [12])

#### 2.5.4.2. Filtro peine (Comb Filter)

En el procesamiento de señales, un filtro peine se produce al sumarle a la señal original una versión retrasada en el tiempo de sí misma, causando así interferencia constructiva y destructiva. La estructura general de un filtro peine es mostrada en la figura 2.35 y esta descrita por:

$$y(n) = x(n) + a \cdot x(n-K) \tag{2.14}$$

donde K es el tamaño del retraso (medido en muestras), y $a$ es un factor de escalamiento aplicado a la señal retrasada.

La función de transferencia del sistema descrito por (2.14) en el dominio de $z$

$$\frac{Y(z)}{X(z)} = \left(1 + a \cdot z^{-K}\right) \tag{2.15}$$

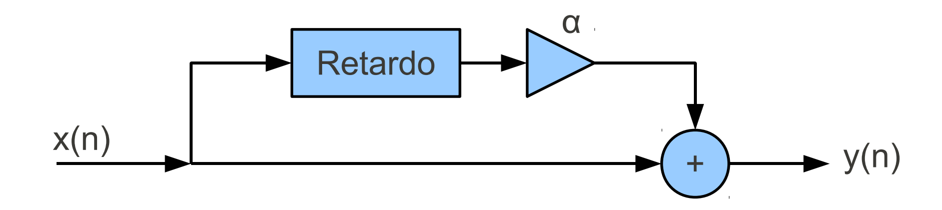



Figura 2.35.: Estructura de un filtro comb FIR tipo Feedforward.

## 2.6. GStreamer

GStreamer es un marco de trabajo *(framework)*, diseñado especialmente para crear aplicaciones multimedia. Esta herramienta fue diseñada especialmente para crear aplicaciones de audio y video; sin embargo es capaz de procesar otros tipos de flujo de información. La API está escrita en C y está basada en GObject y Glib, lo que permite manejo de información orientada a objetos y herencia [24].

Está diseñado para ser utilizado en distintas plataformas como x86, PowerPC, ARM, Solaris, entre otros. Además existen diversas interfaces de GStreamer para varios lenguajes de programación entre ellos destacan Python, C++, Ruby y finalmente Vala, lenguaje utilizado en la realización de la aplicación del proyecto.

La técnica utilizada por *GStreamer* para procesar archivos multimedia consiste en enlazar un conjunto de elementos en una secuencia de reproducción llamada *"pipeline"*. El "elemento", la parte fundamental dentro de la clase de objetos en *GStreamer*, permite crear una cadena de componentes enlazados entre sí y logra que la información fluya a través de ella.

Un elemento tiene funciones específicas como:

- Leer datos de un archivo
- Decodificar los datos
- Enviar datos a una tarjeta de sonido (u otro dispositivo).

Al colocar en una cadena distintos elementos, se pueden realizar tareas específicas, como reproducción o captura multimedia. *GStreamer* provee de una colección de elementos separados en paquetes según su funcionalidad, soporte, y estabilidad. Los elementos pueden ser conectados siempre y cuando sean compatibles, dependiendo de las características que soporten.

A la negociación de características entre elementos se le llama **pads**. Estos son usados para definir los enlaces y el flujo de datos entre elementos de GStreamer [22]. Los "Pads" pueden ser de entrada ("src pads") o de salida ("sink pads"). El *src pad* se encarga de enlazar el archivo, o la fuente de multimedia al flujo de datos (mp3, m4a, mp4, avi, ogg Vorbis, entre otros) y el *sink pad* se encarga de enlazar el flujo de datos al hardware multimedia.

Se pueden definir los "*pipelines*" como contenedores para una colección de elementos. Son una sub-clase de la clase "element", por tanto se pueden manejar como tal. Su utilidad está en, por ejemplo, cambiar el estado de todos los elementos de un flujo de datos cambiando solo el estado de la cadena contenedora [22]. La estructura de la *API* de *GStreamer* tiene una organización de bloques o elementos llamados plugins. Dichos elementos estan separados en cuatro clases:

**gst-plugins-base:** Contiene el conjunto básico de elementos soportados.

**gst-plugins-good:** Contiene el conjunto de elementos soportados que usan licencias preferidas (libres) por los desarrolladores de GStreamer.

**gst-plugins-ugly:** Contiene el conjunto de elementos soportados, pero, que podrían tener problemas para su libre distribución.

**gst-plugins-bad:** Contiene el conjunto de aquellos elementos menos desarrollados que no han pasado las rigurosas pruebas de calidad de los desarrolladores.

Una herramienta para entender cómo funciona *GStreamer* es **gst-launch-version** (por ejemplo gst-launch-0.10). En dado caso que se quiera reproducir un archivo mp3 se podría ejecutar una línea de comando desde la terminal de *GNU/Linux* como se muestra en la figura 2.36.

```
$ gst-launch-0.10 filesrc location="archivo.mp3" !
mad ! alsasink
conjuntoting pipeline to PAUSED ...
Pipeline is PREROLLING ...
Pipeline is PREROLLED ...
conjuntoting pipeline to PLAYING ...
```

Figura 2.36.: Flujo de datos reproducción de audio

El diagrama presentado en la figura 2.37 ejemplifica una secuencia de reproducción (*pipeline*) usando *GStreamer* con los siguientes pasos :

- Se lee el archivo fuente desde un sistema de archivos.
- Se decodifica el archivo fuente : El decodificador, elemento escencial de la secuencia de reproducción, convierte el archivo, normalmente comprimido en mp3 o aac, en muestras de PCM para que puedan ser interpretadas por el elemento sink que puede ser la tarjeta de sonido ALSA .
- ALSA traduce la información en sonido audible que puede ser escuchado por parlantes o audífonos.

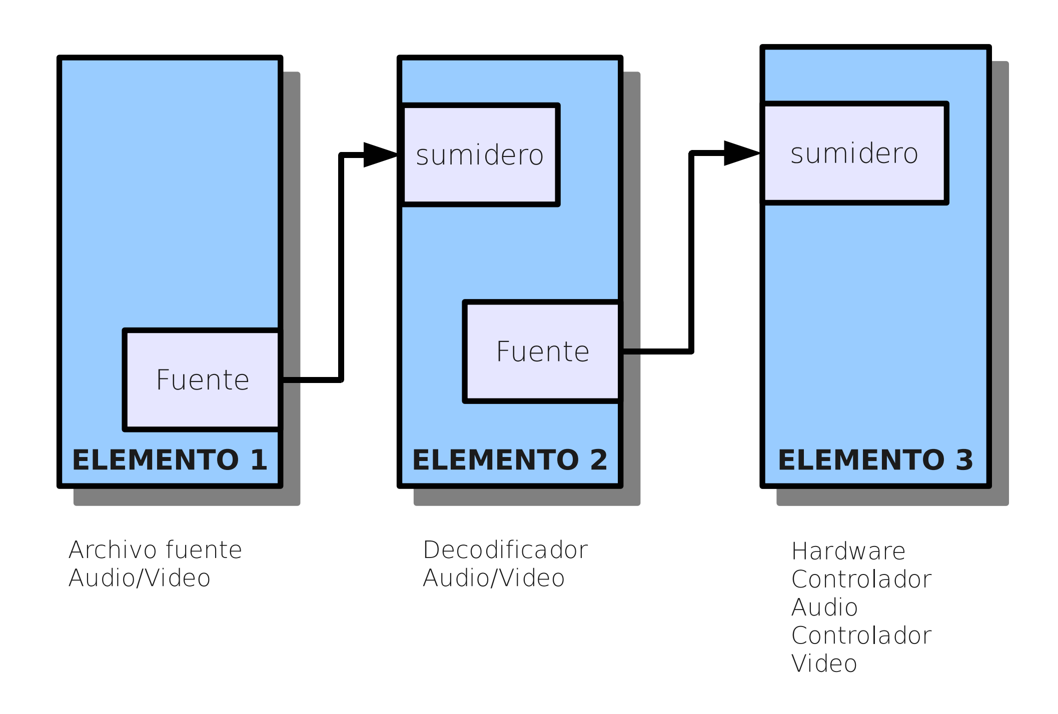



Figura 2.37.: Secuencia básica de reproducción con GStreamer

# 3. Estructura del SDK para la Arquitectura OMAP-L138

## 3.1. El SDK para la tarjeta OMAP-L138 Zoom Evm

La solución planteada es implementar un *SDK* con un esquema similar a los que la empresa maneja para otras plataformas. Específicamente se enfoca en crear un marco de trabajo que se adapte al *SDK* e incorpore las herramientas necesarias para habilitar un método de creación de algoritmos *DSP,* para que se construyan tomando como base la *API* de *iUniversal*.

Se propone un esquema de trabajo en el cual se integra un SDK básico, componente fundamental para el soporte de la tarjeta *OMAP-L138 ZoomEvm*, donde se incluyen los siguientes paquetes de software:

- El gestor de arranque
- El sistema operativo embebido (kernel de GNU/Linux)
- El sistema de archivos
- El sistema de instalación
- Aplicaciones como alsamixer, GStreamer, Qt, D-Bus, etc
- DVSDK

En la figura 3.1 se muestra la estructura de bloques de la solución planteada para el *SDK*. A través de secuencias de comandos se construye un directorio de trabajo*,* con componentes de *software* especializados. Dichos componentes se encuentran almacenados en un repositorio de *subversion* y a través de comandos especializados licenciados por *RidgeRun* llamados *RRTools* se descarga cada uno de ellos.

El diagrama de la figura 3.2 modela la solución desarrollada a nivel de la arquitectura. Se muestra la distribución de las capas de software sobre los dos procesadores de la tarjeta a través de la integración de un sistema operativo con el debido soporte para la arquitectura OMAP-L138.

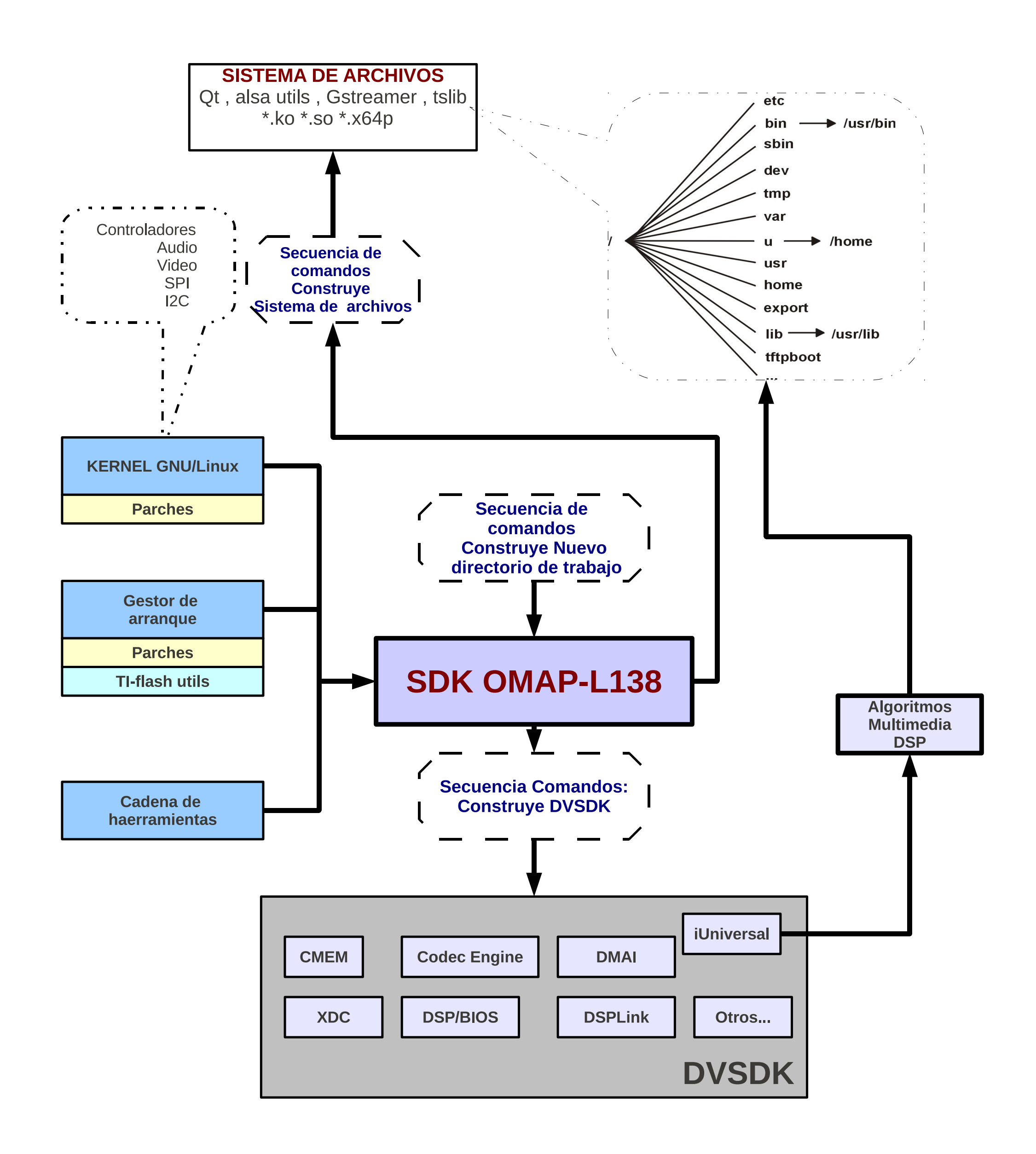



Figura 3.1.: Diagrama generalizado de construcción del SDK

Un *kit* de desarrollo está diseñado para simplificar la implementación de aplicaciones de *software*. Gracias a la división de software, ***DaVinci PSP,*** de *Texas Instruments* existen *kits* con imágenes precompiladas y código fuente del *Kernel* de *GNU/Linux*, así como del gestor de arranque, entre otros componentes, compatibles para las diferentes plataformas que distribuyen.

El núcleo de GNU/Linux utilizado proviene de una rama del kernel *git DaVinci* [13] y el gestor de arranque procede de una rama del *bootloader* de *git U-Boot* [48].

En la integración del *SDK* de la *OMAP-L138 ZoomEvm* se hizo uso del paquete de

*software DaVinci_03_20 03_20_00_08* que se puede obtener en la páginas oficiales de Texas Instruments [29] y contiene los componentes mencionados en la tabla 3.1.

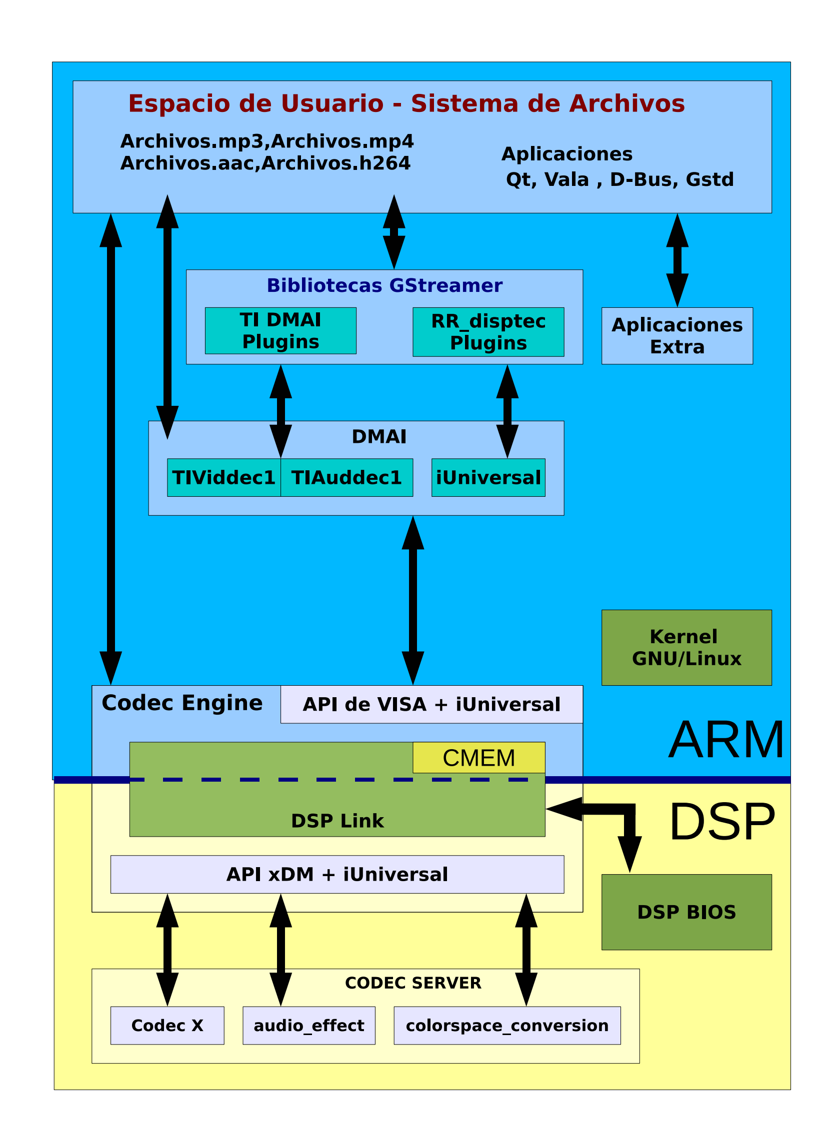



Figura 3.2.: Esquema generalizado de la solución

Tabla 3.1.: Componentes de la versión 03.20.00.08 *DaVinci PSP*

| Componente | Versión | Referencia | Etiqueta |
|---|---|---|---|
| DaVinci Linux Kernel | 2.6.32-rc6 | [13] | d9d2b372d8641f93 720da204a6a756e7 |
| U-Boot | 2009.08 | [48] | v2009.08 |
| ToolChain | ARM 2009q1-203 | [28] | GCC 4.3.3 |

*DaVinci PSP* provee una plataforma básica de desarrollo sin ningún tipo de integración automatizada, donde el usuario es el encargado de portar las imágenes o el código fuente manualmente a la tarjeta de desarrollo. En este proyecto, para realizar la integración con la versión del *kernel 2.6.32* y del *U-Boot*, con la herramienta del *SDK*, se crean parches de software que modifican algunos de sus archivos, para que los menús de configuración propios del kernel y del *U-Boot* se encuentren a disposición en el menú de configuración principal del SDK. Este software al ser especializado para la arquitectura *OMAP-L138*, permite que en los menús de configuración se puedan habilitar los controladores de audio y de video (Ver sección 2.1).

El sistema de archivos implementado permite la opción en el menú de configuración del *SDK* de ser tipo *NFS*, que puede ser accesado por red y uno que puede ser almacenado en una memoria *SD*. Las bibliotecas que se añaden en el sistema de archivos permiten al usuario crear aplicaciones. Dentro de estas bibliotecas destacan: *GStreamer*, *Qt*, *D-Bus*, entre otras.

## 3.2. Integración del DVSDK en el SDK de la OMAP-L138 Zoom Evm

Una vez que los componentes básicos se integran en un paquete unificado, se procede a agregar el software propietario, y con el fin de no modificar el código se realizan pilas de parches, las cuales se encargan de modificar el código sin perder la información de los archivos originales y poder así distribuir el código en otros kit de desarrollo de software empotrado (*SDK*) sin tener la necesidad de almacenar los paquetes propietarios.

El *DVSDK* es una colección de componentes de software integrados para demostrar la interoperabilidad, funcionalidad y desempeño de elementos básicos de *software* en un dispositivo. Además es el paquete que permite la integración de aplicaciones multimedia que puedan hacer uso de interfaces necesarias para ejecutar los algoritmos de DSP.

La compilación de la mayoría de estos paquetes depende de rutas específicas del kernel. Por lo tanto la construcción del *DVSDK* consiste en un proceso de compilación y enlace, donde se deben corregir las rutas de los archivos de cabecera del *kernel* de *GNU/Linux 2.6.32*, que son utilizados por estos paquetes. En esta tarea existe el riesgo que, de una versión a otra, se renombren archivos o incluso se eliminen. De ahí la importancia del registro que se lleva en los sistemas de control de versiones. Entre los paquetes del *DVSDK* a los cuales se les debió realizar modificaciones, almacenados como parches de software están:

- Codec Engine

- DSPLink
- Codec Server
- DMAI

Dentro del paquete de *DSPLink* es donde se generan los módulos de *CMEM* y de *DSPLink*. El paquete de software de *CMEM* (véase sección 2.3.2) permite instalar un módulo denominado "cmem.ko" dentro del *kernel de GNU/Linux* para reservar una sección de memoria contigua, que permite el acceso de datos entre procesadores. Para lograr lo anterior, se deben de especificar los parámetros necesarios en los archivos *Makefile* de este paquete, para utilizar los scripts específicos de configuración para una arquitectura *OMAP-L138*. Además se debe conocer el mapa de memoria RAM de la arquitectura, para definir dónde se instalará la partición de *CMEM*. En la figura 3.3 se muestra las direcciones de memoria virtual de RAM del kernel instalado.

La configuración hecha para el kernel de *GNU/Linux* que se utiliza en la tarjeta, establece que el total de memoria RAM utilizable sea de 64MB, a pesar de que se dispone de una memoria RAM de 128 MB. Con este valor y conociendo la posición de memoria física donde se guarda el kernel en memoria RAM, se puede establecer el espacio correspondiente al módulo de *CMEM*. Para este proyecto, se establece un espacio de 16 MB para *CMEM*. Conociendo las direcciones virtuales de inicio y fin del *kernel de GNU/Linux*, se evita el traslape de este módulo en memoria RAM con el kernel.

El paquete *DSPLink* genera un módulo denominado *"dsplink.ko"* y al igual que el módulo de *CMEM,* se configura a través de los archivos *Makefile* del paquete los scripts dedicados para la arquitectura *OMAP-L138*, que permite reservar un espacio de memoria RAM en tiempo de ejecución para manejar los datos y el código que realiza la comunicación entre procesadores a través del protocolo *IPC*.

El paquete *Codec Engine* no necesita compilarse, pero su integración en el *DVSDK* es necesaria para utilizar su código fuente (*API*) en la construcción tanto de la aplicación que ejecuta los módulos de algoritmos de DSP como en la creación de éstos.

Otro paquete que necesita ser configurado es el *Codec Server*. *Codec Server* genera un archivo ejecutable denominado "servidor" encargado de administrar la memoria dedicada al DSP para el manejo de datos, código, objetos, de los módulos de algoritmos que se integran a este ejecutable. Éste a su vez es almacenado en un espacio determinado de memoria RAM en tiempo de ejecución para su uso. Dentro de este ejecutable, es donde se van a manejar las referencias de los módulos de algoritmos de DSP y es creado a partir de un proceso de compilación, que incluye una serie de archivos y estándares manejados por las *XDCTOOLS* y el compilador de TI. Su construcción va de la mano con la construcción de los módulos de algoritmos de DSP con el marco de referencia *iUniversal*, que se explicará más adelante.

En la figura 3.3 se muestra la segmentación de memoria RAM que se estableció en tiempo de ejecución, gracias a la configuración de *CMEM*, *DSPLink* y *Codec Server*.

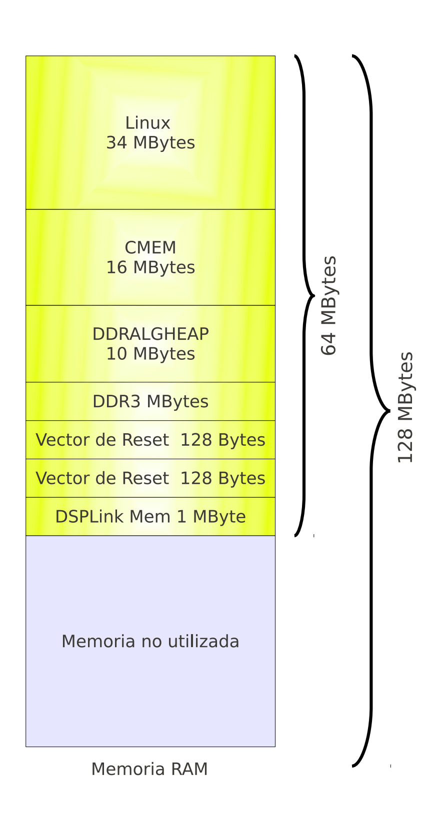



Figura 3.3.: Mapa de Memoria establecido para la arquitectura OMAP-L138

Las particiones referidas por *DDRALGHEAP y DDR* son particiones de memoria dedicadas al DSP. La partición *DDRALGHEAP* contiene toda la memoria dinámica de los algoritmos de DSP que se integran en el *CodecServer,* y la partición *DDR* contiene el código de DSP y los datos estáticos para los algoritmos.

El paquete de *DMAI*, funciona como un conjunto de *API* que facilita el manejo de búferes entre la aplicación (ARM) y el algoritmo de DSP. Es una capa de abstracción que se utiliza en la aplicación para el uso de búferes que permiten el procesamiento de datos a través de las capas de *Codec Engine, DspLink y CodecServe*r. La compilación de este paquete debe ser direccionada a la arquitectura OMAP-L138, que se logra con ciertas modificaciones en sus archivos que también fueron guardados en archivos de parches de software.

En los siguientes cápitulos se explicará el desarrollo de los algoritmos de audio y de video escogidos junto a una descripción acerca de la integración de dichos algoritmos con la interfaz de *iUniversal* y su integración dentro del paquete *CodecServer*. Finalmente se añade una posible solución acerca de cómo se puede desarrollar una aplicación ejecutada en el ARM, a través de *GStreamer*, que utiliza las API de *CodecEngine, DSPLink y CodecServer,* las cuales ayudan a la aplicación a comunicarse con los algoritmos de DSP creados.

# 4. Creación de los módulos multimedia

## 4.1. Módulo de Video

### 4.1.1. Descripción del módulo

El módulo de video propuesto para ejemplificar el uso de la API de *IUniversal* es un algoritmo para la conversión de espacios de color de un flujo de datos de video. Específicamente la conversión del espacio de color I420 (Y:4, Cr:2, Cb:2) a RGB565.

A través de una aplicación en *GStreamer*, y gracias al ambiente que se establece en la tarjeta *OMAP-L138 ZoomEvm* para habilitar el controlador de hardware para el despliegue de video en la pantalla LCD, se puede decodificar, convertir el espacio de color y desplegar en pantalla un archivo de video en formato MPEG4 o H264. Como se muestra en la figura 4.1.

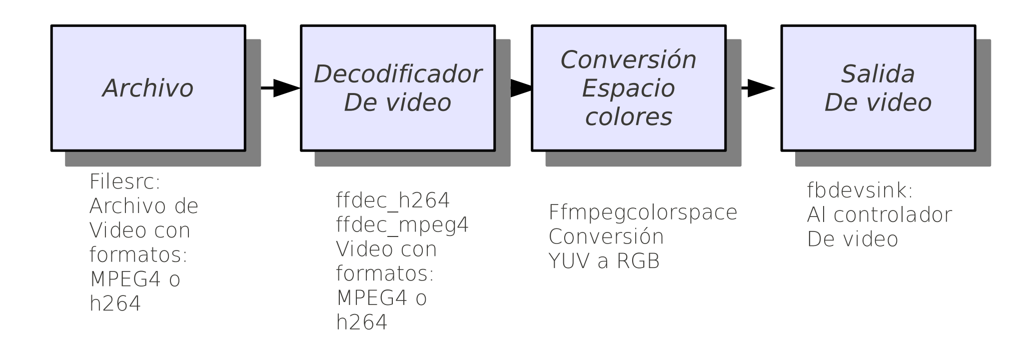



Figura 4.1.: Secuencia de elementos de *GStreamer* utilizado para el algoritmo de video

La secuencia a ejecutar es la siguiente:

- A través de un sistema de archivos enlazado con el kernel de GNU/Linux se lee un archivo con formato mpeg4 o h264.
- Un algoritmo de decodificación descomprime el archivo a datos de video.
- Se pasa por el tranformador de espacio de colores de I420 a RGB565.
- Se pasa el flujo de datos al elemento de GStreamer fbdevsink

○ El elemento fbdevsink se comunica con el módulo del kernel de GNU/Linux que maneja el dispositivo de video LCDC-controller.

Como una forma de utilizar las herramientas integradas en el *SDK* para la arquitectura *OMAP-L138* para integrar algoritmos de DSP con el estándar *iUniversal,* se sustituye el elemento de conversión de espacio de colores que se ejecuta en el ARM, por un elemento que ejecuta un algoritmo conversor de espacio de colores en el DSP.

### 4.1.2. Algoritmo de conversión de espacio de colores

A través del elemento de decodificación de video de *GStreamer,* se pueden obtener búferes de video interpretables como muestra la figura 4.2. Cada búfer representa una imagen (muestra temporal) de video, con un tamaño de W × H píxeles, y con un formato I420.

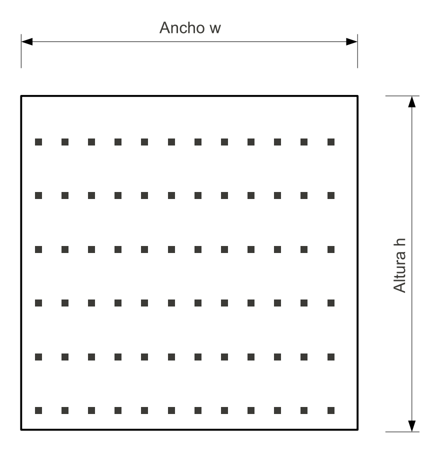



Figura 4.2.: Formato de pixeles para búferes de video de *GStreamer*

El diagrama de la figura 4.3 muestra la conversión a realizar, para pasar un búfer de I420 a RGB565.

El algoritmo que convierte estos espacios debe ejecutar la siguiente ecuación:

$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} c_0 & 0 & c_3 \\ c_0 & c_1 & c_4 \\ c_0 & c_2 & 0 \end{bmatrix} \begin{bmatrix} Y' - 16 \\ Cb - 128 \\ Cr - 128 \end{bmatrix} \tag{4.1}$$

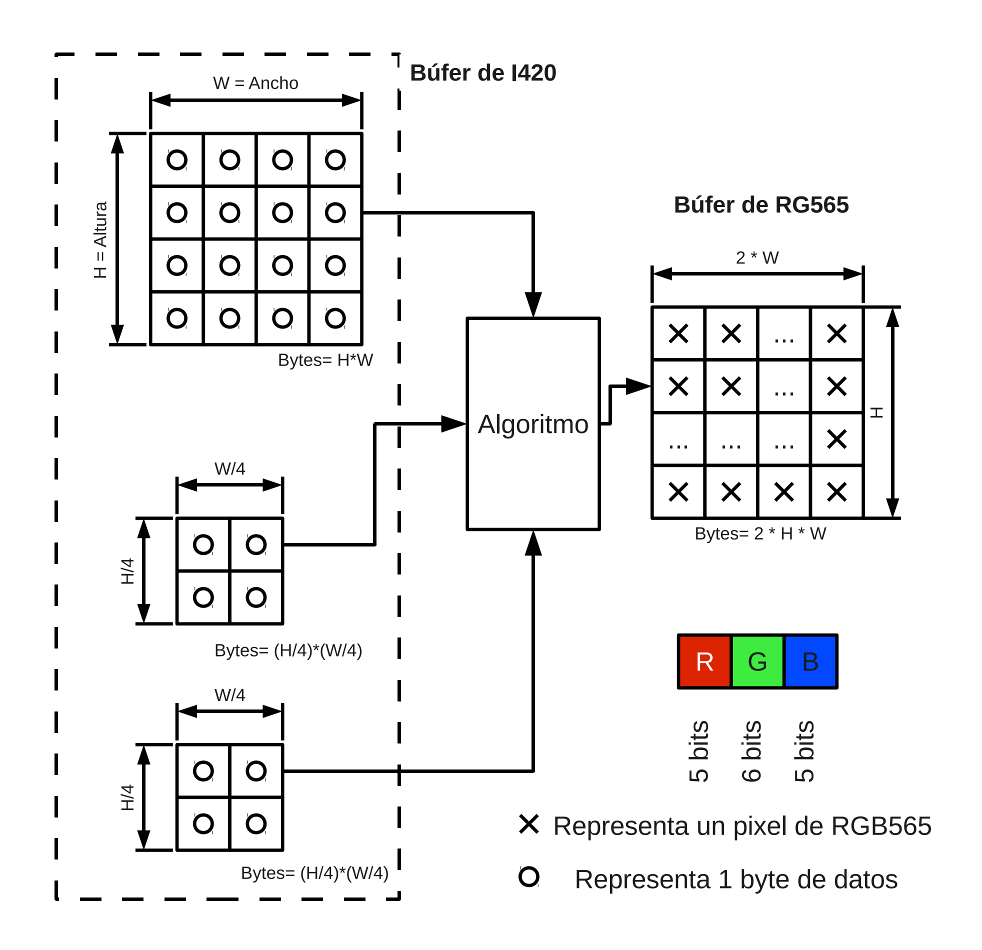



Figura 4.3.: Diagrama de conversión de formato YUV a RGB565

La ecuación (4.1) es utilizada para resolver la conversión de formatos, y está implementada dentro de la biblioteca "*imglib*" [37] (véase licencia en el apéndice A.4) que se encuentra en código C, precompilada para DSP y desarrollada por Texas Instruments. La función *IMG_ycbcr422p_rgb565* de dicha biblioteca permite convertir 2 pixeles de YCbCr 4:2:2 a 2 pixeles RGB565.

El formato I420 es de tipo YCbCr 4:2:0, la única diferencia entre YCbCr 4:2:2 y YCbCr 4:2:0, es que el primero tiene el doble de muestras de crominancia que el segundo, por lo que la biblioteca "*imglib*" se puede utilizar con la modificación de que, por cada dos filas de muestras Y se puede utilizar la misma fila de muestras de crominancias para realizar la conversión.

Los valores correspondientes a los coeficientes en (4.1) son:

- $c_0 = 1{,}0000$
- $c_1 = -0{,}3365$
- $c_2 = 1{,}7324$
- $c_3 = 1{,}3707$

· $c_4 = -0{,}6982$

Esta conversión espera valores enteros de Y en los rangos de 16 a 235, y valores enteros de Cb y Cr entre 16 y 240, que es lo que el decodificador da en su salida. El decodificador otorga valores enteros con rangos de corrimiento en las muestras de Y con un valor de 16 y en las muestras de Cb y Cr con un valor de 128.

Cuando se recibe un búfer de una imágen de video de I420 éste es procesado por el algoritmo de DSP para obtener uno de RGB565 como se ilustra en la figura 4.4.

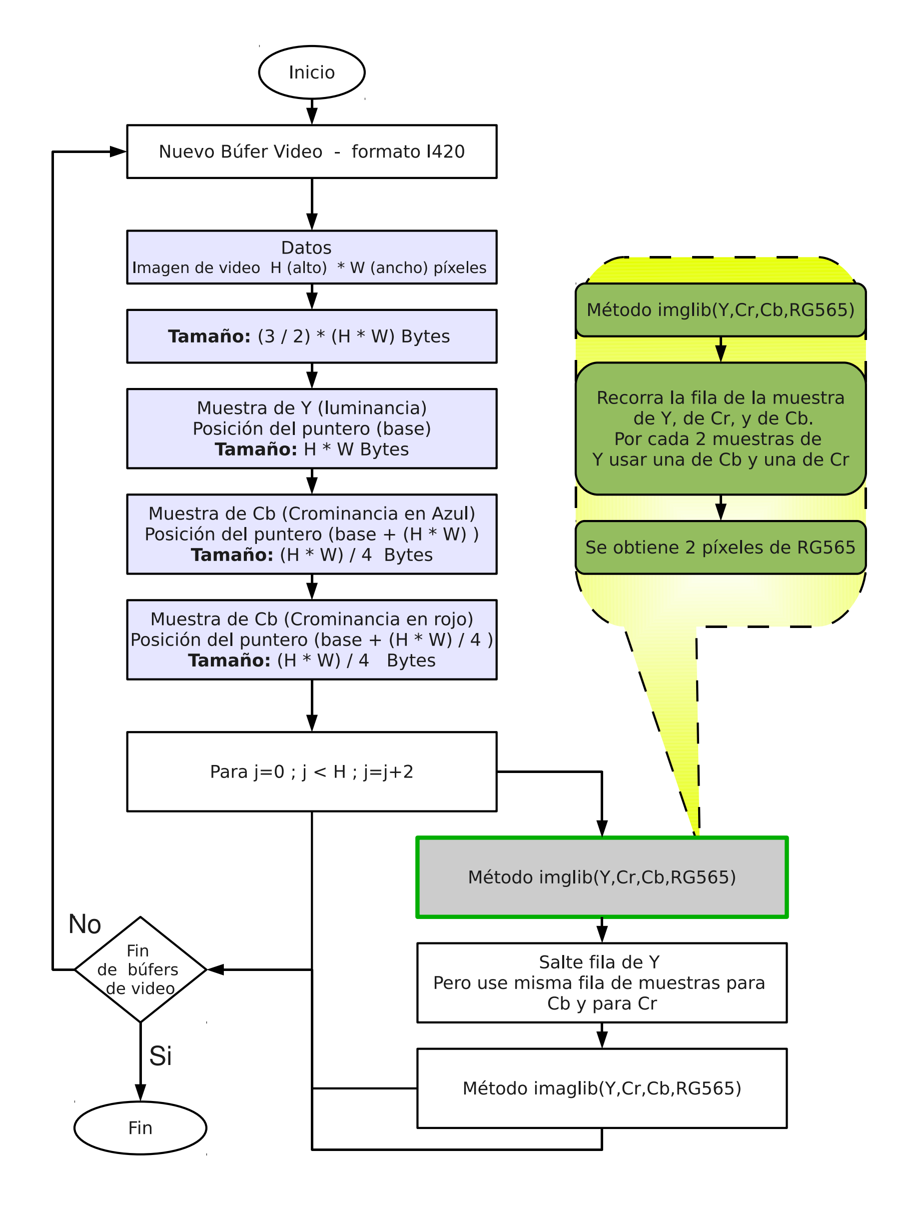



Figura 4.4.: Diagrama funcional del algoritmo de conversión I420 a RGB565

### 4.1.3. Integración del módulo de video con el marco de *iUniversal*

Para poder utilizar el algoritmo de conversión de colores en el DSP se debe enmascarar la implementación anterior con el marco de *IUniversal*. En la figura 4.5 se puede observar un resumen del módulo del algoritmo de video implementado con los principales elementos y funciones del estándar *iUniversal* (refiérase a [34]).

**Objeto:**
**UNIVERSALCSC_TI_Obj *csc // Variable objeto que permite manejar una referencia a un algoritmo de este tipo**
**Búfers de trabajo:**
**XDM1_BufDesc *inBufs; //Buffer de I420**
**XDM1_BufDesc *outBufs; //Buffer de RGB565**
**Parámetros del objeto:**
**ICSC_Params ICSC_PARAMS = //Coeficientes de la ecuación para convertir I420 a RGB565**
**{ sizeof(ICSC_Params),**
**8192,//Coeficiente 1**
**11229, //Coeficiente 2**
**-2757, //Coeficiente 3**
**-5720, //Coeficiente 4**
**14192, //Coeficiente 5**
**320, //Ancho**
**240 //Altura };**

**Cálculos:**
**Función I420 a RGB565: IMG_yuv420pl_to_rgb565 (Y puntero , Cr puntero, Cb puntero, RGB puntero, valor ancho, valor de altura)**

$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} 1{,}0000 & 0{,}0000 & 1{,}3707 \\ 1{,}0000 & -0{,}3365 & -0{,}6982 \\ 1{,}0000 & 1{,}7324 & 0{,}0000 \end{bmatrix} \times \begin{bmatrix} Y' - 16 \\ Cb - 128 \\ Cr - 128 \end{bmatrix}$$

**Asignación de memoria:**
**Bufer de entrada: Memoria estática contigua de tamaño del Bufer de I420**
**Bufer de salida: Memoria estática contigua de tamaño del Bufer de RGB565**
**Funciones Necesarias del estándar *iUniversal*:**
**UNIVERSALCSC_TI_alloc : Realiza la petición de memoria para el algoritmo.**
**UNIVERSALCSC_TI_free : Libera la memoria del algoritmo y la referencia al algoritmo.**
**UNIVERSALCSC_TI_initObj : Carga al objeto del algoritmo los valores estáticos iniciales que necesite durante su ejecución, como los coeficientes para la ecuación de conversión de espacio de colores. Esta función se comunica con el marco de *CodecEngine* cada vez que se necesite crear un algoritmo de este tipo.**
**UNIVERSALCSC_TI_process : Esta función se comunica con el marco de *CodecEngine* cada vez que se necesite ejecutar la función de conversión de I420 a RGB565 (IMG_yuv420pl_to_rgb565).**
**UNIVERSALCSC_TI_control :**

Figura 4.5.: Marco de implementación del algoritmo de conversión de I420 a RGB565

En la figura 4.6 se puede observar la estructura del directorio que se genera cuando se crea el módulo de algoritmo *xDM* a través de la estructura de paquetes utilizadas con las *XDCTOOLS,* que permiten luego utilizar este paquete dentro del proceso de creación del *codecServer,* el cual genera el archivo ejecutable que contiene el módulo de conversión de

espacio de colores y que puede ser llamado por una aplicación en el ARM a través del marco de *CodecEngine* cuando se requiera ejecutar el algoritmo.

```
|-- UNIVERSAL_CSC.version.0.0.1.wizardversion.0.5.3
|-- UNIVERSAL_CSC.xdc
|-- universal_csc.c
|-- universal_csc_ti.h
|-- universal_csc_ti_priv.h
|-- ce
|        -- UNIVERSAL_CSC.xdc
|        |-- UNIVERSAL_CSC.xs |
|        |-- config.bld|
|        |-- package.bld |
|        |-- package.mak |
|        |-- package.xdc
|-- lib
|        | -- universal_csc.a64P--> archivo que se
va a generar
|-- link.xdt
|-- package.bld
|-- package.xdc
|-- package.xs

|-- csc.c // Archivo que invoca la librería de TI
```

Figura 4.6.: Estructura del codec de video

Para una explicación más detallada de cómo construir un módulo de estos, para luego integrarlo dentro de un *codecServer,* se puede observar el manual técnico de la sección A.1.

## 4.2. Módulo de Audio

### 4.2.1. Implementación del módulo

Se ha seleccionado para ejemplificar un algoritmo de audio procesado en el DSP de la OMAP-L138 la implementación de un filtro FIR tipo peine con la ecuación de diferencias

$$y(n) = G_1x(n) + G_2x(n-d_1) + G_3x(n-2d_1) + G_4x(n-3d_1) \tag{4.2}$$

- $d_1 = 1152$
- $G_1 = 1/2$
- $G_2 = 1/4$
- $G_3 = 1/4$
- $G_4 = 1/4$

El búfer de entrada está compuesto por una secuencia de muestras

$$x(n) = \{x_1, x_2 \ldots, x_{n-1}, x_n\} \tag{4.3}$$

El búfer de salida esta compuesto por una secuencia de muestras

$$y(n) = \{y_1, y_2 \ldots, y_{n-1}, y_n\} \tag{4.4}$$

Finalmente luego de aplicar los coeficientes se puede observar la implementación del filtro

$$y(n) = \frac{1}{2} \cdot x(n) + \frac{1}{4} \cdot x(n-1152) + \frac{1}{4} \cdot x(n-2304) + \frac{1}{4} \cdot x(n-3456) \tag{4.5}$$

La función de transferencia del filtro en el plano de $z$ esta dado por (4.6)

$$\frac{Y(z)}{X(z)} = \frac{1}{2} + \frac{1}{4} \cdot z^{-1152} + \frac{1}{4} \cdot z^{-2304} + \frac{1}{4} \cdot z^{-3456} \tag{4.6}$$

Se puede expresar como :

$$\frac{Y(z)}{X(z)} = \frac{1}{4} (1 + (a+1)(a+j)(a-j))$$

con

$$a = z^{-1152}$$

El filtro expresado en (4.6) suma la entrada con la respuesta de la cascada de los tres filtros peine, cada uno con 1152 ceros sobre un círculo unitario. El primer filtro peine esta desfasado $\pi/1152$ y el segundo en $-\pi/2304$ y el último en $\pi/2304$.

- El tamaño del búfer de entrada es igual al tamaño del búfer de salida.
- La frecuencia de muestreo es $f_s = 44100Hz$
- $n$ es el número de muestras por búfer y se obtiene de acuerdo a los cálculos presentados en la sección 2.33 a partir de (2.9).

La figura 4.7 esquematiza el diagrama de bloques que ejemplifica la implementación del algoritmo presentado en (4.2).

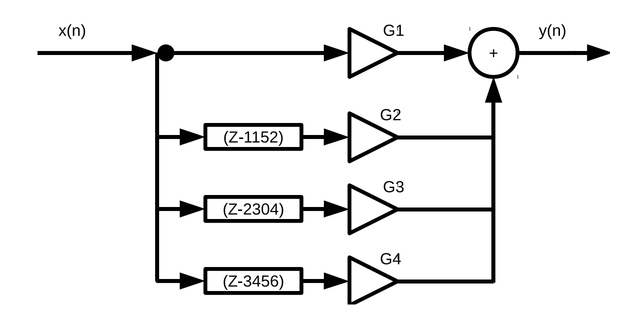



Figura 4.7.: Implementación del filtro tipo peine

### 4.2.2. Manejo de los búferes

Se implementa un manejo de los búferes como se muestra en la figura 4.9, donde se almacenan 3 bloques de datos para poder realizar (4.5).

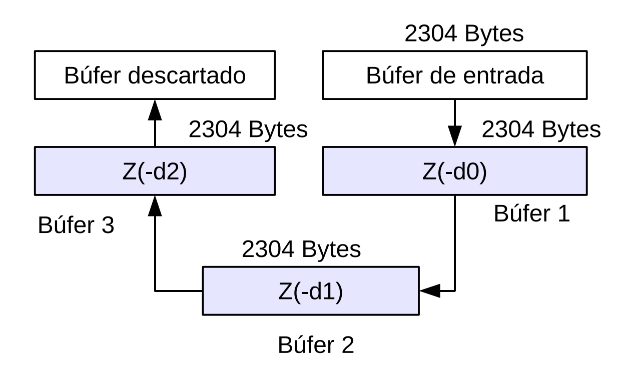



Figura 4.8.: Secuencia de copia de los búferes

Cada uno de los búferes contiene 2304 bytes que representan 1152 muestras, como se puede observar en la figura 4.9.

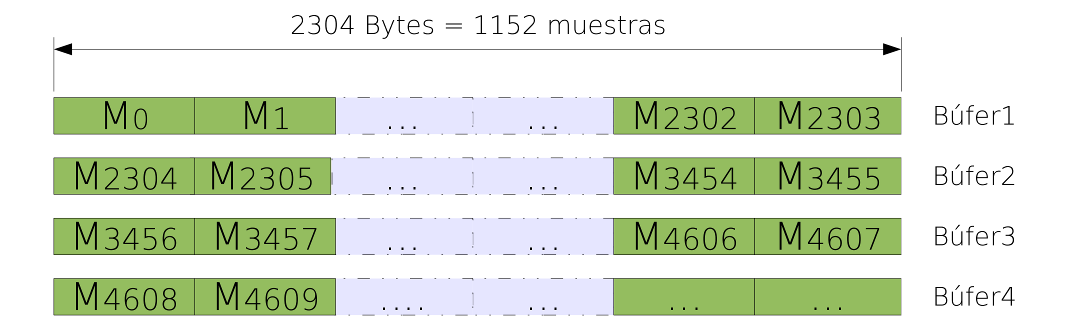



Figura 4.9.: Manejo de los búferes

### 4.2.3. Representación numérica

La representación de punto fijo utiliza un número fijo de dígitos para representar el entero y las fracciones de los números reales. Se usa la notación $Q.i.f$ para representar una variable de tamaño de punto fijo de $i+f$, $i$ representa los dígitos utilizados para la parte entera y $f$ dígitos utilizados para representar la parte fraccionaria.

Los principales obstáculos asociados con la aritmética de punto fijo son el desbordamiento máximo y el desbordamiento mínimo. Para el desarrollo de módulo de audio se utiliza la notacion $Q$24,8. Dicho formato de punto fijo utiliza 24 bits de número entero y 8 bits de decimal.

| Representación entera | Representación decimal |
|---|---|
| 0xh000000 | 0xh00 |

Como el flujo de datos se trabaja con muestras de 16 bits se debe de convertir cada muestra a la representación $Q$24,8, y para lograrlo, se realiza un corrimiento de 8 posiciones. Utilizando el siguiente ejemplo se puede ilustrar cómo se realiza la transformación de datos en formato 16 bits a formato $Q$24,8.

| valor | | corrimiento | | valor entero | valor decimal |
|---|---|---|---|---|---|
| 0h0EA0 | « | 8 | => | 0h0EA0 | 00 |

### 4.2.4. Diagrama de flujo del algoritmo

En la figura 4.10 se muestra el diagrama del algoritmo de audio. La ejecución del algoritmo tiene 4 etapas que sobresalen: la carga de los parámetros, la transformación de las muestras, el procesamiento de los datos y finalmente el corrimiento en el arreglo de búfers.

En la primera etapa , los parámetros son recibidos por el algoritmo provenientes del elemento de GStreamer, donde son definidos por el usuario. En la transformación de las muestras , los datos son cambiados del formato $Int16$ a un formato punto fijo $Q24,8$. Esta tranformación se da con el fin de no perder precisión en los cálculos y evitar desbordamientos de los datos.

Una vez que los cálculos se completan, el dato de salida debe ser convertido nuevamente a un formato de $Int16$ , para que pueda ser retornado a la aplicación. El lazo se repite hasta que la cantidad de búferes retorne nulo o bien se termine la ejecución del archivo de audio.

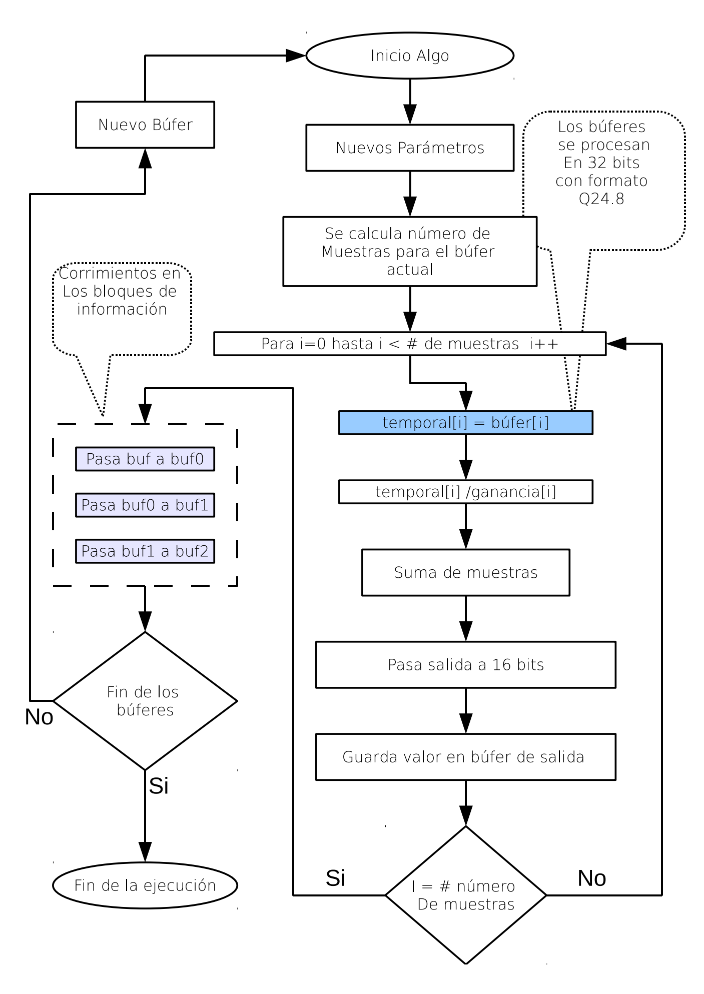



Figura 4.10.: Diagrama de flujo del algoritmo de audio

### 4.2.5. Integración del módulo de audio con el marco de *iUniversal*

Para poder utilizar el módulo implementado en el DSP se debe enmascarar la implementación anterior con el marco de *IUniversal*. En la figura 4.11 se puede observar una

recapitulación de las principales elementos y funciones del estándar *iUniversal* (refiérase a [34]) para el módulo del algoritmo de audio.

**Objeto:**
**AUDIO_EFFECT_TI_Obj *audio_effect // Variable objeto que permite manejar una referencia a un algoritmo de este tipo**
**Búfers de trabajo:**
**XDM1_BufDesc *inBufs; //Buffer de entrada**
**XDM1_BufDesc *outBufs; //Buffer de salida**
**XDAS_Int32 búfer0**
**XDAS_Int32 búfer1**
**XDAS_Int32 búfer2**
**Parámetros del objeto:**

**Cálculos:**

$$y(n) = \frac{1}{2} \cdot x(n) + \frac{1}{4} \cdot x(n-1152) + \frac{1}{4} \cdot x(n-2304) + \frac{1}{4} \cdot x(n-3456)$$

**Asignación de memoria:**
**Bloque de memoria del búfer de entrada pasa a ser el bloque de memoria búfer0**
**Bloque de memoria búfer0 pasa a ser bloque de memoria búfer1**
**Bloque de memoria búfer1 pasa a ser bloque de memoria búfer2**
**Funciones Necesarias del estándar *iUniversal*:**
**AUDIO_EFFECT_TI_alloc : Realiza la petición de memoria para el algoritmo.**
**AUDIO_EFFECT_TI_free : Libera la memoria del algoritmo y la referencia al algoritmo.**
**AUDIO_EFFECT_TI_initObj : Carga al objeto del algoritmo los valores estáticos iniciales que necesite durante su ejecución. Esta función se comunica con el marco de *CodecEngine* cada vez que se necesite crear una instancia del algoritmo.**
**AUDIO_EFFECT_TI_process : Esta función se comunica con el marco de *CodecEngine* cada vez que se necesite ejecutar el algoritmo**

Figura 4.11.: Marco de implementación del algoritmo de audio: filtro FIR

En la figura 4.12 se puede observar la estructura del directorio que se genera cuando se crea el módulo de algoritmo *xDM* a través de la estructura de paquetes utilizadas con las *XDCTOOLS,* que permiten luego utilizar este paquete dentro del proceso de creación del *codecServer,* el cual genera el archivo ejecutable que contiene el módulo de audio y que puede ser llamado por una aplicación en el ARM a través del marco de *CodecEngine* cuando se requiera ejecutar el algoritmo.

```
.
|-- AUDIO_EFFECT.version.1.0.0.wizardversion.0.5.3
|-- AUDIO_EFFECT.xdc
|-- audio_effect.c
|-- audio_effect_ti.h
|-- audio_effect_ti_priv.h
|-- ce
|        |-- AUDIO_EFFECT.xdc
|        |-- AUDIO_EFFECT.xs
|        |-- package.bld
|        |-- package.mak
|        `-- package.xdc
|-- lib
|        |-- (audio_effect.a64P)--> archivo que se
va a generar
|-- link.xdt
|-- package.bld
|-- package.xdc
|-- package.xs
|-- audio_algorithm.c
`-- audio_algorithm.h
```

Figura 4.12.: Estructura del codec de audio

Para una explicación más detallada de cómo construir un módulo para DSP, para luego integrarlo dentro de un *codecServer,* se puede observar el manual técnico de la sección A.1.

# 5. Aplicación basada en GStreamer

## 5.1. Aplicación general

La aplicación de referencia está basada en el paquete de *GStreamer* (véase sección 2.6) y está escrita en *Vala*.

Como se observa en la figura 5.1, la aplicación de *GStreamer* es un enmascaramiento de las capas inferiores de software dentro de las cuales destacan *XDAIS*, *iUniversal*, *CodecEngine, DSPLink*, entre otros.

Los sumideros (sinks) reciben los datos desde las fuentes (sources) de otros elementos. El elemento creado se encarga de codificar la información de tal modo que los paquetes (*codecs* o algoritmos) puedan procesarla. Los búfers de ***CMEM***, son el flujo de información especializada para producir cadenas de datos codificadas y en conjunto con *DSPLink* funcionan como el canal de información entre la aplicación y el módulo DSP (véase figura 5.1).

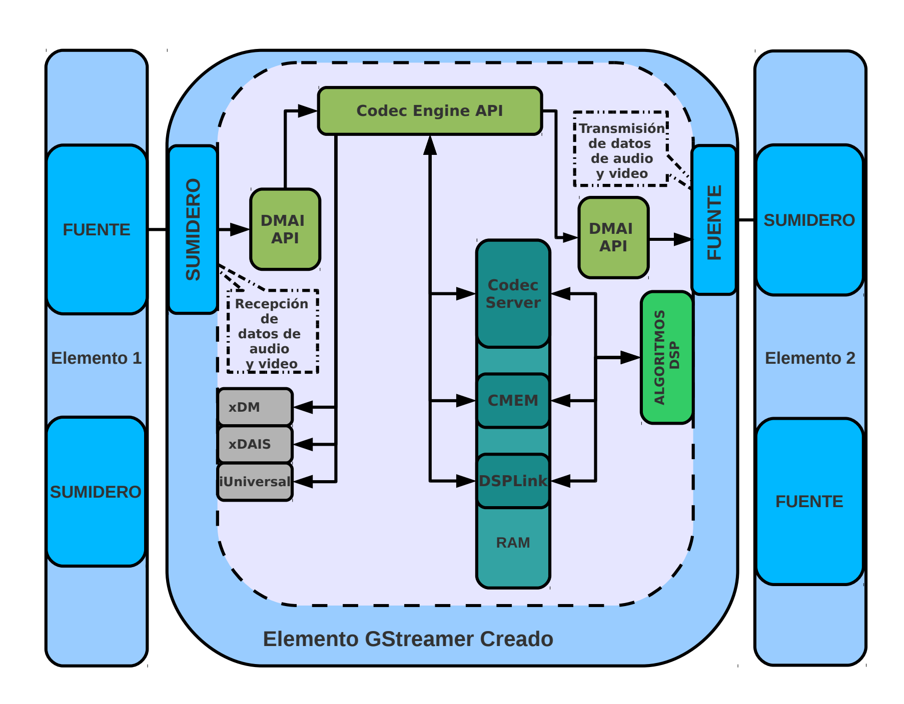



Figura 5.1.: Aplicación de *GStreamer* (Tomado de [30])

Una vez que la información ha sido codificada se envía al módulo, el cual es encargado de llamar las funciones que se ejecutan en el DSP y gracias a los búferes de salida de *CMEM*, el flujo de datos procesado es retornado al elemento de *GStreamer* donde el bloque fuente (src) se encarga de pasar la información al siguiente elemento. La figura 5.2 presenta el diagrama de flujo referente al elemento de *GStreamer* creado.

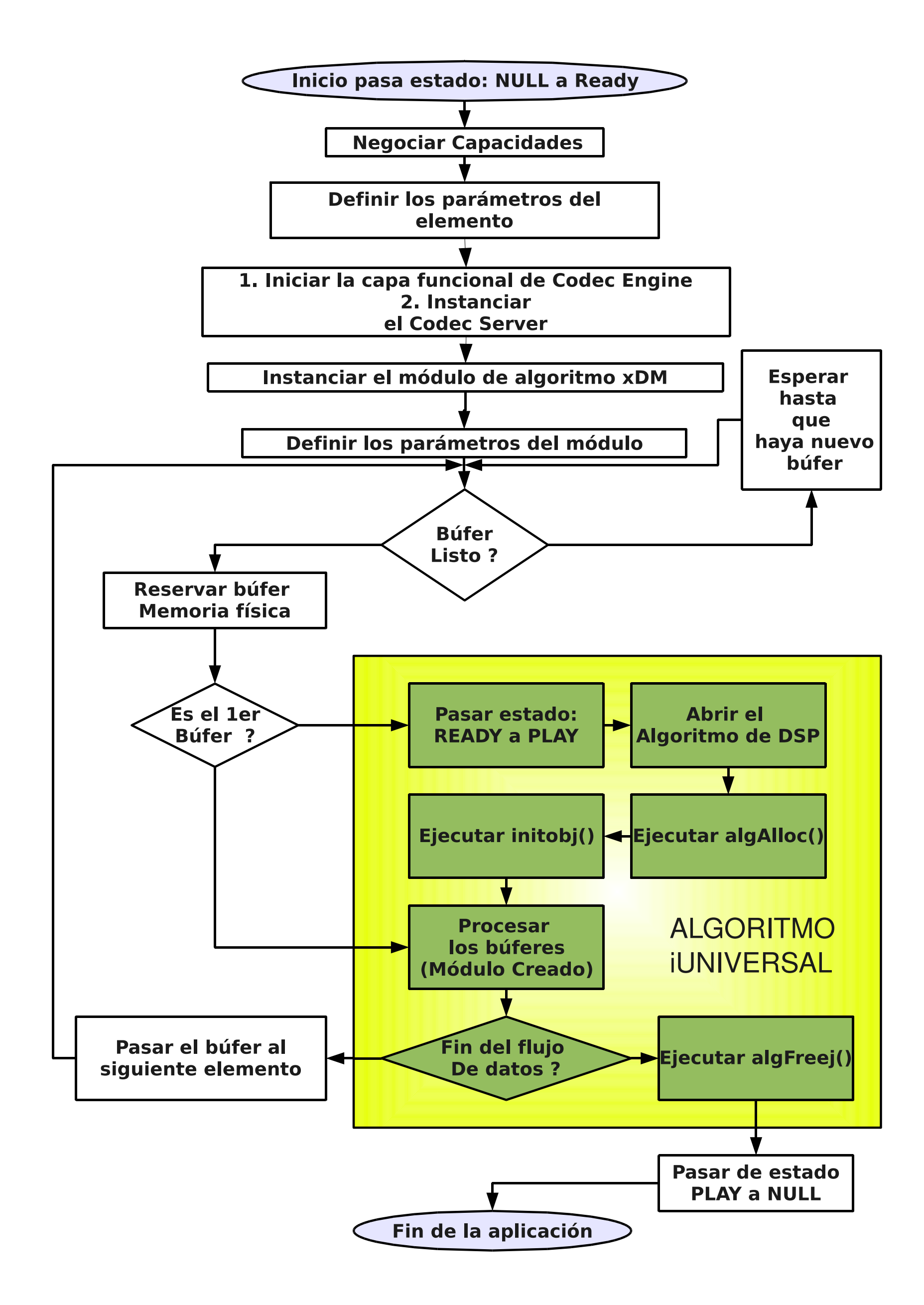



Figura 5.2.: Diagrama de flujo de la aplicación de gstreamer

En la figura 5.2 se ilustran las etapas que deben de cumplirse en la ejecución de una aplicación de *GStreamer*. Además se mencionan las funciones que conforman la ejecución de un módulo con algoritmos DSP.

## 5.2. Flujo de datos de GStreamer utilizado para el módulo de audio

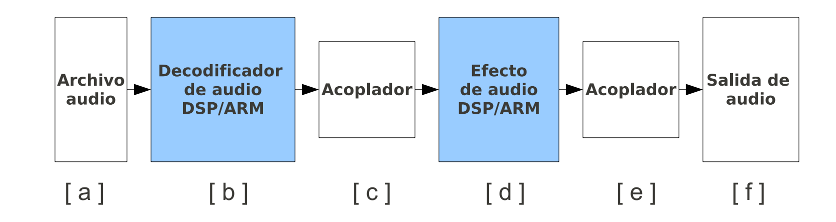



Figura 5.3.: Secuencia de elementos de *GStreamer* utilizado para el algoritmo de audio

Como se observa en la figura 5.3, el flujo de datos de *GStreamer*, contiene seis elementos:

- El elemento [a] representa el archivo fuente, este puede tener dos tipos de formatos mp3 o aac
- El decodificador de audio mostrado en el elemento [b], puede variar, entre los decodificadores compatibles se encuentran :
  - mad : Decodificador de mp3 ejecutado en el procesador ARM, es parte del paquete **gst-plugins-good**.
  - faad: Decodificador de aac ejecutado en el procesador ARM, es parte del paquete **gst-plugins-bad**.
  - TIAuddec1: Decodificador de aac ejecutado en el procesador DSP.
- Los elementos [c] y [e] representan los acoples al módulo, estos son necesarios ya que las capacidades que puede recibir el módulo son limitadas y en caso que no sean compatibles deben de transformarse.
  - El elemento se llama audioconvert.
- El elemento [d] representa el algoritmo y recibe como parámetros G1,G2,G3,G4. Estos sirven para modificar la atenuación en los cálculos de las muestras.
- Finalmente el elemento [f] representa la capa del hardware, es el encargado de hacer la conexión entre GStreamer y el controlador de audio de la tarjeta.

# 6. Análisis y Resultados

## 6.1. Resultados de Audio

En la figura 6.1 se muestra una gráfica del audio en función del tiempo generado con el software GNU/Octave. La gráfica superior representa el audio original y la gráfica inferior representa el audio modificado por el módulo algoritmo DSP creado. En la tabla 6.1 se muestra la información referente al flujo de GStreamer utilizado para obtener el audio presentado en la figura 6.1.

Tabla 6.1.: Características de la muestra de audio utilizada con audio modificado

| **Variables** | **Características** |
|---|---|
| Codificación | Con signo 16 bits PCM. |
| Ordenamiento | Little-endian. |
| Canales | 2 canales estéreo. |
| Frecuencia de muestreo | 44100 Hz. |
| Decodificador de audio | mad (GPP) |
| Efecto de audio | dt_audioeffect (DSP) |
| Formato de audio | mp3 |
| Flujo de datos de GStreamer utilizado | gst-launch filesrc location= Vertigo_high.mp3 ! mad ! audioconvert ! dt_audioeffect g1=2 g2=4 g3=4 g4=4 ! filesink audio-modificado.wav |

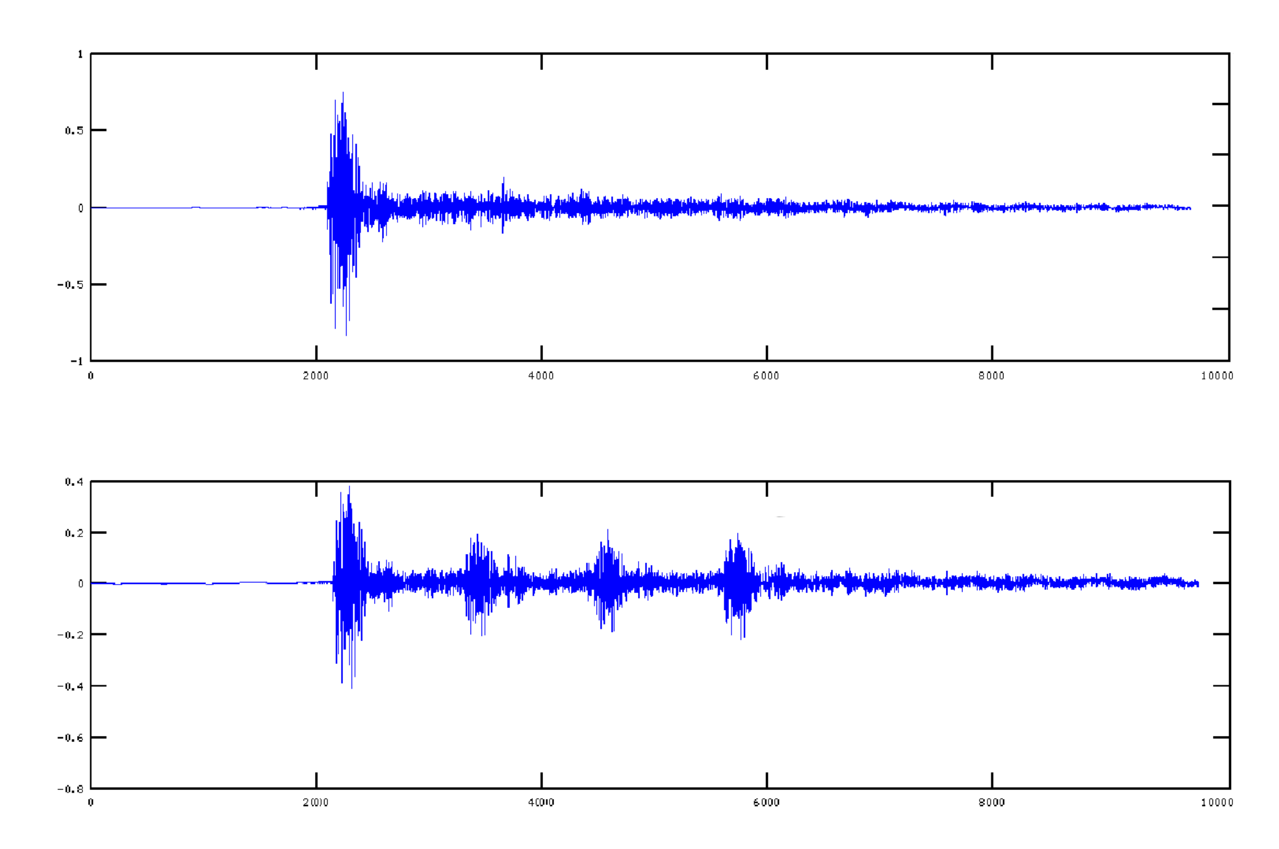



Figura 6.1.: Audio resultante del proceso DSP

Se muestra en la figura 6.1 el resultado de procesar el audio a través del módulo creado para DSP. El filtro FIR implementado debía adicionar en una señal resultante muestras anteriores, en este caso se muestra como se reproduce la señal en tres ocaciones.

A continuación se presenta una comparación en el consumo de GPP que tienen los flujos de datos de GStreamer usando combinaciones de elementos de GStreamer que son procesados en el GPP y en el DSP. En la tabla 6.2 se muestran los elementos de GStreamer que se ejecutan en el GPP y los elementos creados específicamente para el procesamiento en el DSP. Durante la ejecución de los archivos AAC se hace uso de un paquete DSP que no fue creado para el proyecto sino que se reutiliza con el fin de comparar el consumo de GPP con más de un elemento de GStreamer siendo ejecutado en el DSP; se trata del elemento *TIAuddec1*.

Tabla 6.2.: Elementos de GStreamer utilizados para las pruebas de rendimiento

| **Elementos GPP** | **Elementos DSP** | **Función** |
|---|---|---|
| filesrc | - | Abrir una fuente o archivo |
| ***faad*** | ***TIAuddec1*** | Decodificador de audio en formato aac |
| mad | - | Decodificador de audio en formato mp3 |
| ***audioecho*** | ***dt_audioeffect*** | efecto de audio |
| audioconvert | - | elemento encargado de ajutar las capacidades |
| alsasink | - | Elemento que accesa el controlador de audio |

Se analizan diversos escenarios con el fin de evidenciar el rendimiento de una aplicación, basada en GStreamer, en función del consumo del GPP usando elementos con procesamiento en el DSP y sus homólogos en el procesador GPP. Dentro de los escenarios destacan pruebas con formatos de audio AAC y MP3 con las combinaciones mostradas en la tabla 6.3.

Tabla 6.3.: Procesador en el cual se ejecutan los procesos

| Caso | Formato | Decodificador | Efecto de audio |
|---|---|---|---|
| Caso 1 | ACC | GPP | GPP |
| Caso 2 | | GPP | DSP |
| Caso 3 | | DSP | GPP |
| Caso 4 | | DSP | DSP |
| Caso 5 | MP3 | GPP | GPP |
| Caso 6 | | GPP | DSP |

La tabla 6.4 muestra los datos tomados para la reproducción de un mismo archivo de audio con una duración de 70 segundos y tomando una muestra de consumo de GPP cada segundo.

La figura 6.2 es la representación gráfica para la tabla 6.4, y seguidamente se detalla su representación.

Table 6.4.: Consumo de GPP para los casos de audio

| Tiempo (s) | caso 1 (%) | caso 2 (%) | caso 3 (%) | caso 4 (%) | caso 5 (%) | caso 6 (%) |
|---|---|---|---|---|---|---|
| 1 | 98 | 91 | 92 | 27 | 91 | 75 |
| 2 | 100 | 70 | 93 | 14 | 76 | 51 |
| 3 | 100 | 54 | 95 | 17 | 76 | 56 |
| 4 | 101 | 51 | 90 | 6 | 77 | 49 |
| 5 | 100 | 65 | 91 | 0 | 76 | 55 |
| 6 | 99 | 69 | 94 | 1 | 77 | 48 |
| 7 | 100 | 52 | 94 | 0 | 76 | 44 |
| 8 | 100 | 53 | 93 | 0 | 76 | 42 |
| 9 | 100 | 60 | 93 | 1 | 77 | 46 |
| 10 | 100 | 53 | 94 | 0 | 77 | 44 |
| 11 | 100 | 62 | 96 | 0 | 76 | 43 |
| 12 | 100 | 65 | 97 | 0 | 77 | 44 |
| 13 | 100 | 65 | 88 | 1 | 77 | 43 |
| 14 | 101 | 77 | 88 | 1 | 77 | 43 |
| 15 | 99 | 86 | 96 | 0 | 76 | 42 |
| 16 | 100 | 79 | 93 | 0 | 76 | 43 |
| 17 | 100 | 77 | 94 | 1 | 76 | 42 |
| 18 | 100 | 89 | 97 | 0 | 76 | 44 |
| 19 | 100 | 92 | 95 | 0 | 76 | 42 |
| 20 | 100 | 93 | 92 | 0 | 77 | 42 |
| 21 | 100 | 88 | 95 | 2 | 76 | 43 |
| 22 | 100 | 89 | 91 | 0 | 76 | 41 |
| 23 | 100 | 93 | 95 | 0 | 76 | 41 |
| 24 | 100 | 90 | 93 | 1 | 76 | 40 |
| 25 | 100 | 92 | 96 | 1 | 76 | 42 |
| 26 | 100 | 90 | 93 | 0 | 76 | 44 |
| 27 | 100 | 90 | 94 | 0 | 76 | 45 |
| 28 | 100 | 91 | 94 | 0 | 76 | 44 |
| 29 | 100 | 93 | 91 | 0 | 77 | 44 |
| 30 | 100 | 88 | 95 | 1 | 77 | 43 |

| Tiempo (s) | caso 1 (%) | caso 2 (%) | caso 3 (%) | caso 4 (%) | caso 5 (%) | caso 6 (%) |
|---|---|---|---|---|---|---|
| 31 | 100 | 91 | 94 | 1 | 78 | 44 |
| 32 | 99 | 92 | 94 | 1 | 77 | 43 |
| 33 | 100 | 91 | 93 | 1 | 77 | 45 |
| 34 | 99 | 91 | 94 | 1 | 77 | 43 |
| 35 | 100 | 91 | 92 | 3 | 77 | 43 |
| 36 | 100 | 92 | 93 | 3 | 77 | 44 |
| 37 | 100 | 94 | 93 | 8 | 77 | 42 |
| 38 | 100 | 94 | 90 | 4 | 77 | 42 |
| 39 | 100 | 93 | 93 | 14 | 77 | 41 |
| 40 | 99 | 94 | 93 | 13 | 78 | 43 |
| 41 | 100 | 94 | 95 | 22 | 78 | 45 |
| 42 | 99 | 91 | 98 | 38 | 79 | 46 |
| 43 | 101 | 94 | 93 | 42 | 78 | 45 |
| 44 | 99 | 94 | 92 | 27 | 78 | 45 |
| 45 | 100 | 94 | 95 | 39 | 78 | 44 |
| 46 | 100 | 95 | 94 | 38 | 78 | 45 |
| 47 | 100 | 94 | 92 | 37 | 78 | 44 |
| 48 | 100 | 92 | 90 | 42 | 78 | 46 |
| 49 | 100 | 96 | 92 | 38 | 78 | 44 |
| 50 | 100 | 94 | 95 | 39 | 78 | 44 |
| 51 | 100 | 91 | 92 | 47 | 79 | 45 |
| 52 | 101 | 93 | 95 | 44 | 79 | 43 |
| 53 | 100 | 93 | 92 | 41 | 80 | 43 |
| 54 | 99 | 92 | 90 | 38 | 79 | 42 |
| 55 | 100 | 92 | 91 | 42 | 79 | 44 |
| 56 | 100 | 93 | 94 | 38 | 79 | 46 |
| 57 | 100 | 92 | 93 | 38 | 79 | 47 |
| 58 | 100 | 95 | 95 | 40 | 79 | 46 |
| 59 | 100 | 91 | 92 | 38 | 79 | 46 |
| 60 | 100 | 95 | 96 | 43 | 79 | 45 |
| 61 | 100 | 96 | 93 | 41 | 80 | 46 |
| 62 | 100 | 95 | 91 | 45 | 81 | 45 |
| 63 | 100 | 93 | 92 | 47 | 80 | 47 |
| 64 | 100 | 97 | 95 | 46 | 80 | 45 |
| 65 | 101 | 95 | 93 | 42 | 80 | 46 |
| 66 | 100 | 95 | 91 | 38 | 80 | 44 |
| 67 | 100 | 96 | 93 | 39 | 80 | 44 |
| 68 | 100 | 95 | 96 | 47 | 80 | 43 |
| 69 | 100 | 93 | 93 | 44 | 80 | 45 |
| 70 | 100 | 92 | 90 | 38 | 79 | 46 |
| **Promedio** | **99,93** | **86,31** | **93,20** | **18,87** | **77,84** | **44,87** |

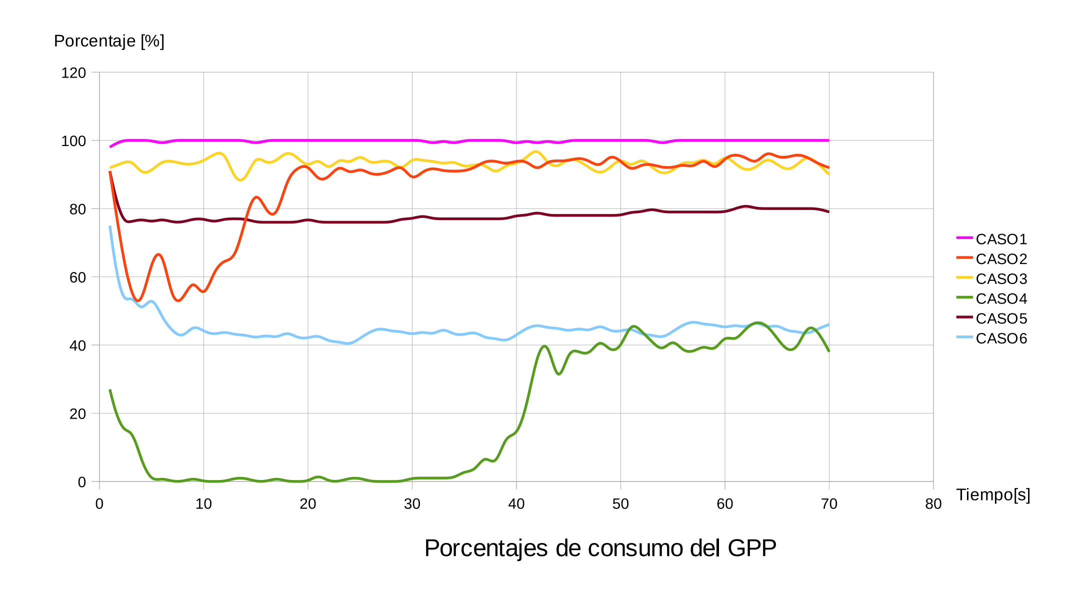



Figura 6.2.: Comparación del consumo de GPP en los casos de audio

La figura 6.3 es la representación gráfica para el consumo del recursos en el procesador DSP, se muestran los casos 2 y 4 donde destaca módulos de algoritmos que se ejecutan en el procesador antes mencionado. Sus valores promedio se pueden observar en la tabla 6.5.

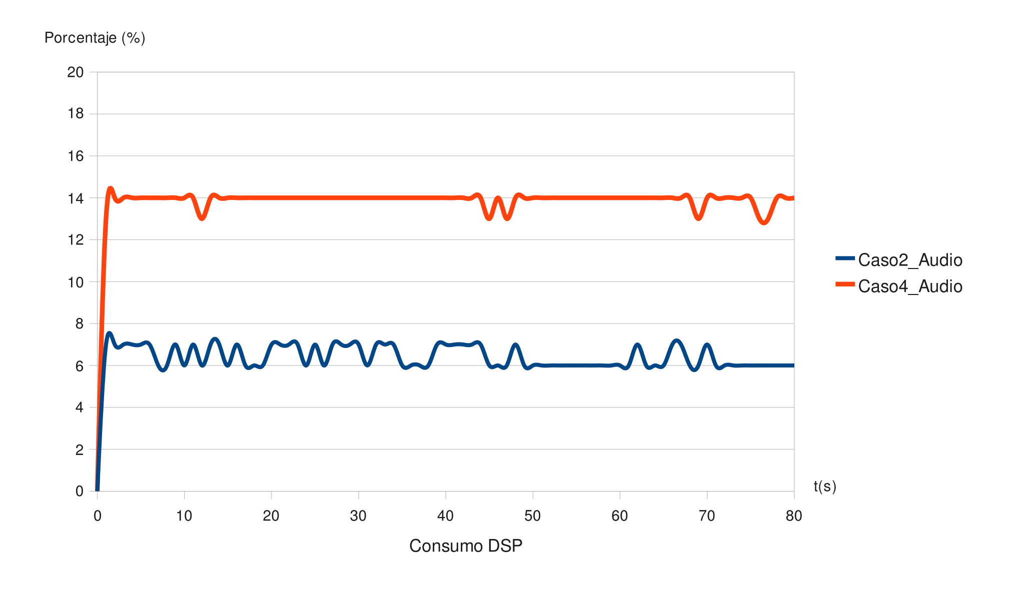



Figura 6.3.: Consumo de DSP para los casos 2 y 4

Se puede observar de los resultados con respecto al consumo de recursos en el DSP mostrados en la figura 6.3 que éstos no sobrepasan el 14 % para el caso extremo en el cual ambos módulos de algoritmos se ejecutan en el DSP.

Tabla 6.5.: Tabla Resumen de consumo de recursos para archivos AAC

| Caso | Consumo de GPP [ %] | Mejora [ %] | Consumo de DSP [ %] | Elementos en el DSP |
|---|---|---|---|---|
| caso 1 | 99,93 | - | - | - |
| caso 2 | 86,31 | 13,26 | 6,4 | Decoficador |
| caso 3 | 93,20 | 6,73 | - | Filtro |
| caso 4 | 18,87 | 81,12 | 13.9 | Decodificador y filtro |

El rendimiento con respecto a la utilización de GPP para los casos de reproducción de archivos en formato AAC se puede observar en la figura 6.2 , y la información detallada con respecto al flujo de datos de *GStreamer* utilizados se pueden observar en los apéndices A.3.1.1, A.3.1.2, A.3.1.3 y A.3.1.4.

Cuando se ejecuta un solo elemento DSP en el flujo de *GStreamer* la mejora en el consumo de GPP se encuentra entre 6,73 % y 13,26 %. El elemento de decodificación de audio presenta un mejor rendimiento en un flujo de reproducción de *GStreamer* ya que tiene un porcentaje de consumo de GPP menor, equivalente a 86,31 % . Dicho resultado se puede observar en la tabla comparativa 6.5. Para los escenarios de reproducción de audio en formato AAC destaca una mejora del 81,12 % cuando el elemento de decodificación y el elemento que describe el efecto de audio se ejecutan en el DSP, dichos resultados se muestran en la tabla 6.5.

Tabla 6.6.: Tabla Resumen de consumo de GPP para archivos AAC

| Caso | Consumo de GPP [ %] | Mejora [ %] | Elementos DSP |
|---|---|---|---|
| caso 5 | 77,84 | - | 0 |
| caso 6 | 44,87 | 42,36 | 1 |

El rendimiento con respecto a la utilización de GPP para los casos de reproducción de archivos en formato MP3 se puede observar en la figura 6.2 , y la información detallada con respecto al flujo de datos de *GStreamer* utilizados se pueden observar en los apéndices A.3.1.5 y A.3.1.6.

Los escenarios de reproducción de audio en formato MP3 se limitan a hacer uso del efecto de audio ejecutado en el DSP, ya que no se dispone de un decodificador de MP3 que se

ejecute en el DSP. Al limitarse la cantidad de pruebas se debe de hacer uso solamente del decodificador de MP3 por defecto de GStreamer: mad.

Para los casos 5 y 6 se presenta una mejora en el porcentaje de consumo de GPP con respecto a la reproducción de los archivos con formato AAC. Cuando ambos elementos (decodficador y efecto) se ejecutan en el GPP detaca un consumo del 77,84, un valor 22,1 % más eficiente que el valor presentado en su caso homólogo para la reproducción de AAC (caso 1). Cuando se agrega un elemento de GStreamer que se ejecuta en el DSP existe una mejora del 42,36 % con respecto al caso anterior (caso 5).

Los casos anteriores demuestran que la integración implementada con las herramientas del *SDK* para la creación de algoritmos de *DSP* con respecto a audio permite obtener una mejora en el consumo de *GPP*. Una optimización de los algoritmos ya existentes para el *GPP* en el *DSP* llevaría a obtener un menor consumo del *GPP* y por ende un mejor aprovechamiento del procesador GPP, lo que favorece la ejecución de procesos simultáneos en la plataforma.

## 6.2. Resultados de Video

Una forma de medir el rendimiento para los casos de video es comparar el consumo de GPP cuando se ejecuta un flujo de datos de *GStreamer*. Se hace uso de algoritmos de decodificación y de conversión de espacio colores para el despliegue de video en el GPP, y elementos con función similar que integran algoritmos de DSP. Los resultados son comparaciones entre líneas de procesos que utilizan elementos propios de *GStreamer* y los elementos creados en este proyecto. En la tabla 6.7 se encuentran las contra-partes de los elementos que son de *GStreamer* y se ejecutan en el GPP y los elementos creados específicamente para el procesamiento en el DSP. El elemento para archivos de h264 o mpeg4 de *GStreamer* que ejecuta el algoritmo para decodificación en el DSP, se denomina *TIViddec2*. Este elemento forma parte de un paquete de Texas Instruments. El elemento que se creó en este proyecto se denomina *dt_colorspace*.

Tabla 6.7.: Elementos de GStreamer utilizados para las pruebas de rendimiento

| Elementos GPP | Elementos DSP | Función |
|---|---|---|
| filesrc | - | Abrir una fuente o archivo |
| ffmpegcolorspace | dt_colorspace | Conversión de espacio de colores I420 a RGB565 |
| ffdec_h264 | TIViddec2 | Decodificar archivos formato h264 |
| ffdec_mpeg4 | TIViddec2 | Decodificador de archivos de mpeg4 |
| qtdemux | - | demultiplexor de señales de archivos en containers como mp4 |
| fbdevsink | - | Elemento que accesa el controlador de video |

A continuación se presenta una serie de pruebas que permiten observar el comportamiento con respecto al consumo de GPP y cuadros por segundo desplegadas en una secuencia de video para diferentes escenarios de ejecución. Para todos los casos se generan gráficas que representan el consumo del GPP durante la ejecución de la aplicación y otra que representa el número de imágenes o cuadros por segundo que se despliega en pantalla, y para los casos 2 y 3 específicamente se tomó el consumo de DSP. Todas las pruebas se realizaron con muestras tomadas cada un segundo en un lapso de 80 segundos por prueba.

Tabla 6.10.: Tabla Resumen de consumo de recursos para los casos de video 1,2,3

| Caso | Consumo de GPP [%] | Mejora [%] | Consumo de DSP [%] | Elementos DSP |
|---|---|---|---|---|
| caso 1 | 99,9 | - | - | - |
| caso 2 | 84,9 | 13,26 | 22,71 | Decodificador |
| caso 3 | 24 | 6,73 | 35,74 | Decodificador y Algoritmo de Video |

Para obtener estos datos se utilizó un elemento de *GStreamer* que permite acceder a los datos de calendarización y consumo del GPP (ARM), llamado *dmaiperf*, que también permite obtener el flujo de datos de video que se están transmitiendo en la salida video. Estos se pueden observar en las tablas 6.8 y 6.9, respectivamente. Dentro de los escenarios destacan pruebas con formatos de video *h264* y *mpeg4*, además de un caso con un contenedor *mp4*.

Los casos 1, 2 y 3 utilizan un archivo de video con formato h264 con contenedor mp4. Se utiliza un demultiplexador para obtener el flujo de datos de video en h264, además se hace uso de un elemento de decodificación de h264 y conversión de espacio de colores. El caso 1 ejecuta ambos elementos en el procesador GPP. El caso 2 ejecuta el algoritmo decodificador en en el DSP y el algoritmo convertidor de espacio de colores en el GPP. Por último, el caso 3 ejecuta ambos algoritmos en el DSP.

El rendimiento con respecto a la utilización de GPP y el despliegue de cuadros por segundo, para estos casos se puede observar en la figuras 6.4 y 6.5 respectivamente, y la información detallada con respecto al flujo de datos de *GStreamer* utilizados se pueden observar en los apéndices A.3.2.1, A.3.2.2 y A.3.2.3.

Table 6.8.: Consumo de GPP para video

| Tiempo (s) | caso 1 (%) | caso 2 (%) | caso 3 (%) | caso 4 (%) | caso 5 (%) | caso 6 (%) | caso 7 (%) |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 96 | 105 | 122 | 103 | 111 | 101 | 73 |
| 2 | 101 | 103 | 39 | 99 | 120 | 99 | 16 |
| 3 | 97 | 103 | 15 | 100 | 100 | 100 | 13 |
| 4 | 96 | 103 | 14 | 100 | 32 | 100 | 22 |
| 5 | 100 | 100 | 13 | 100 | 17 | 100 | 15 |
| 6 | 101 | 98 | 17 | 100 | 19 | 100 | 17 |
| 7 | 98 | 95 | 17 | 100 | 13 | 100 | 15 |
| 8 | 99 | 96 | 15 | 100 | 15 | 100 | 15 |
| 9 | 102 | 75 | 11 | 100 | 12 | 100 | 18 |
| 10 | 99 | 77 | 14 | 100 | 14 | 100 | 13 |
| 11 | 100 | 76 | 27 | 101 | 16 | 100 | 31 |
| 12 | 101 | 75 | 22 | 99 | 14 | 100 | 23 |
| 13 | 100 | 77 | 32 | 101 | 17 | 99 | 19 |
| 14 | 100 | 90 | 22 | 100 | 14 | 100 | 25 |
| 15 | 100 | 80 | 25 | 99 | 18 | 100 | 26 |
| 16 | 99 | 85 | 20 | 100 | 15 | 102 | 23 |
| 17 | 100 | 91 | 44 | 100 | 18 | 101 | 23 |
| 18 | 100 | 84 | 63 | 101 | 18 | 101 | 29 |
| 19 | 102 | 86 | 11 | 100 | 18 | 100 | 22 |
| 20 | 100 | 82 | 10 | 100 | 23 | 100 | 19 |
| 21 | 100 | 87 | 13 | 100 | 19 | 100 | 20 |
| 22 | 100 | 106 | 15 | 100 | 19 | 100 | 21 |
| 23 | 100 | 104 | 15 | 100 | 20 | 100 | 21 |
| 24 | 100 | 104 | 14 | 100 | 18 | 100 | 20 |
| 25 | 100 | 81 | 21 | 101 | 18 | 100 | 18 |
| 26 | 100 | 75 | 26 | 99 | 20 | 101 | 17 |
| 27 | 99 | 76 | 28 | 100 | 18 | 99 | 19 |
| 28 | 100 | 77 | 21 | 100 | 19 | 100 | 21 |
| 29 | 102 | 74 | 21 | 100 | 24 | 100 | 32 |
| 30 | 100 | 75 | 24 | 99 | 18 | 100 | 26 |
| 31 | 100 | 84 | 22 | 100 | 17 | 100 | 25 |
| 32 | 100 | 87 | 60 | 100 | 21 | 100 | 20 |
| 33 | 102 | 86 | 37 | 100 | 23 | 100 | 14 |
| 34 | 101 | 78 | 16 | 100 | 24 | 100 | 15 |
| 35 | 98 | 86 | 14 | 100 | 20 | 100 | 14 |
| 36 | 101 | 83 | 17 | 100 | 24 | 100 | 19 |

| Tiempo (s) | caso 1 (%) | caso 2 (%) | caso 3 (%) | caso 4 (%) | caso 5 (%) | caso 6 (%) | caso 7 (%) |
|---|---|---|---|---|---|---|---|
| 37 | 100 | 80 | 15 | 99 | 18 | 100 | 19 |
| 38 | 100 | 84 | 12 | 100 | 25 | 100 | 19 |
| 39 | 101 | 84 | 16 | 100 | 26 | 100 | 17 |
| 40 | 100 | 90 | 16 | 100 | 25 | 100 | 13 |
| 41 | 101 | 106 | 19 | 101 | 21 | 99 | 16 |
| 42 | 99 | 105 | 29 | 100 | 28 | 100 | 17 |
| 43 | 100 | 97 | 26 | 100 | 20 | 99 | 15 |
| 44 | 103 | 76 | 32 | 100 | 21 | 100 | 15 |
| 45 | 99 | 78 | 19 | 100 | 22 | 100 | 18 |
| 46 | 100 | 74 | 31 | 100 | 19 | 100 | 15 |
| 47 | 100 | 78 | 25 | 100 | 15 | 100 | 16 |
| 48 | 100 | 74 | 25 | 101 | 21 | 100 | 19 |
| 49 | 103 | 76 | 21 | 99 | 49 | 100 | 16 |
| 50 | 100 | 75 | 103 | 100 | 116 | 100 | 13 |
| 51 | 101 | 75 | 17 | 99 | 97 | 99 | 22 |
| 52 | 101 | 82 | 15 | 100 | 32 | 100 | 15 |
| 53 | 98 | 78 | 16 | 100 | 12 | 100 | 17 |
| 54 | 99 | 85 | 17 | 100 | 17 | 99 | 15 |
| 55 | 100 | 83 | 15 | 99 | 12 | 100 | 15 |
| 56 | 98 | 82 | 14 | 100 | 18 | 100 | 18 |
| 57 | 100 | 81 | 14 | 100 | 17 | 100 | 13 |
| 58 | 100 | 83 | 18 | 100 | 13 | 100 | 31 |
| 59 | 100 | 80 | 10 | 100 | 19 | 100 | 23 |
| 60 | 100 | 84 | 27 | 100 | 22 | 99 | 19 |
| 61 | 100 | 84 | 25 | 100 | 32 | 100 | 25 |
| 62 | 100 | 95 | 24 | 99 | 15 | 100 | 26 |
| 63 | 102 | 100 | 22 | 100 | 20 | 100 | 23 |
| 64 | 100 | 101 | 21 | 100 | 15 | 100 | 23 |
| 65 | 101 | 84 | 26 | 100 | 12 | 100 | 29 |
| 66 | 100 | 78 | 19 | 100 | 19 | 100 | 22 |
| 67 | 100 | 78 | 22 | 100 | 16 | 100 | 19 |
| 68 | 100 | 78 | 86 | 100 | 13 | 100 | 20 |
| 69 | 101 | 77 | 37 | 100 | 16 | 100 | 21 |
| 70 | 99 | 76 | 15 | 100 | 19 | 100 | 21 |
| 71 | 100 | 76 | 15 | 100 | 23 | 100 | 20 |
| 72 | 100 | 76 | 12 | 100 | 22 | 100 | 18 |
| 73 | 100 | 76 | 13 | 100 | 20 | 99 | 17 |
| 74 | 100 | 83 | 11 | 100 | 19 | 100 | 19 |
| 75 | 99 | 81 | 10 | 100 | 19 | 100 | 21 |
| 76 | 100 | 81 | 9 | 100 | 16 | 100 | 32 |
| 77 | 100 | 81 | 14 | 98 | 17 | 100 | 26 |
| 78 | 100 | 82 | 21 | 101 | 25 | 99 | 25 |
| Promedio | 99.9 | 84.9 | 24 | 99.9 | 25.4 | 99.9 | 20.5 |

Table 6.9.: Cuadros por segundo para los casos de video

| Tiempo (s) | caso1 (fps) | caso2 (fps) | caso3 (fps) | caso4 (fps) | caso5 (fps) | caso6 (fps) | caso7 (fps) |
|---|---|---|---|---|---|---|---|
| 1 | 18 | 20 | 23 | 14 | 21 | 17 | 24 |
| 2 | 26 | 21 | 24 | 13 | 20 | 14 | 23 |
| 3 | 26 | 21 | 24 | 12 | 21 | 13 | 24 |
| 4 | 21 | 21 | 23 | 11 | 24 | 15 | 24 |
| 5 | 10 | 19 | 24 | 10 | 23 | 15 | 24 |
| 6 | 9 | 18 | 23 | 10 | 22 | 15 | 23 |
| 7 | 8 | 18 | 24 | 8 | 22 | 14 | 24 |
| 8 | 8 | 18 | 24 | 8 | 22 | 14 | 24 |
| 9 | 8 | 18 | 24 | 7 | 21 | 14 | 24 |
| 10 | 9 | 18 | 24 | 7 | 21 | 14 | 23 |
| 11 | 9 | 18 | 24 | 7 | 21 | 15 | 24 |
| 12 | 8 | 18 | 24 | 7 | 22 | 15 | 24 |
| 13 | 9 | 18 | 24 | 6 | 22 | 15 | 24 |
| 14 | 9 | 18 | 24 | 6 | 22 | 15 | 23 |
| 15 | 9 | 18 | 24 | 6 | 22 | 15 | 23 |
| 16 | 9 | 18 | 24 | 6 | 23 | 15 | 24 |
| 17 | 9 | 18 | 23 | 7 | 22 | 15 | 24 |
| 18 | 9 | 18 | 24 | 6 | 21 | 15 | 23 |
| 19 | 9 | 18 | 24 | 6 | 21 | 15 | 24 |
| 20 | 9 | 18 | 24 | 6 | 21 | 15 | 24 |
| 21 | 9 | 18 | 24 | 6 | 22 | 15 | 23 |
| 22 | 9 | 18 | 24 | 6 | 23 | 15 | 24 |
| 23 | 9 | 18 | 24 | 6 | 23 | 15 | 23 |
| 24 | 9 | 18 | 23 | 7 | 23 | 14 | 24 |
| 25 | 9 | 18 | 24 | 6 | 22 | 15 | 24 |
| 26 | 9 | 18 | 24 | 6 | 22 | 14 | 23 |
| 27 | 9 | 19 | 23 | 6 | 22 | 15 | 24 |
| 28 | 9 | 18 | 23 | 6 | 23 | 15 | 23 |
| 29 | 9 | 18 | 23 | 7 | 23 | 15 | 23 |
| 30 | 9 | 18 | 24 | 7 | 22 | 15 | 24 |
| 31 | 9 | 18 | 24 | 7 | 22 | 15 | 24 |
| 32 | 9 | 18 | 24 | 6 | 23 | 15 | 24 |
| 33 | 9 | 18 | 24 | 6 | 22 | 16 | 23 |
| 34 | 9 | 18 | 24 | 6 | 21 | 16 | 24 |
| 35 | 9 | 18 | 24 | 6 | 22 | 16 | 23 |
| 36 | 9 | 18 | 24 | 6 | 22 | 16 | 24 |

| Tiempo (s) | caso1 (fps) | caso2 (fps) | caso3 (fps) | caso4 (fps) | caso5 (fps) | caso6 (fps) | caso7 (fps) |
|---|---|---|---|---|---|---|---|
| 37 | 9 | 19 | 24 | 6 | 22 | 16 | 23 |
| 38 | 9 | 19 | 24 | 6 | 23 | 15 | 24 |
| 39 | 9 | 19 | 24 | 7 | 23 | 16 | 24 |
| 40 | 9 | 19 | 24 | 8 | 22 | 16 | 24 |
| 41 | 9 | 18 | 23 | 8 | 22 | 16 | 24 |
| 42 | 9 | 18 | 24 | 8 | 20 | 15 | 24 |
| 43 | 9 | 18 | 24 | 7 | 23 | 14 | 24 |
| 44 | 9 | 19 | 24 | 7 | 22 | 13 | 23 |
| 45 | 9 | 19 | 24 | 7 | 24 | 12 | 24 |
| 46 | 9 | 19 | 24 | 6 | 25 | 12 | 24 |
| 47 | 9 | 19 | 24 | 6 | 24 | 13 | 24 |
| 48 | 9 | 19 | 23 | 6 | 24 | 13 | 23 |
| 49 | 9 | 19 | 24 | 7 | 23 | 12 | 23 |
| 50 | 9 | 19 | 24 | 7 | 23 | 12 | 24 |
| 51 | 9 | 19 | 24 | 6 | 23 | 12 | 24 |
| 52 | 9 | 19 | 23 | 6 | 24 | 11 | 23 |
| 53 | 9 | 19 | 24 | 7 | 24 | 12 | 24 |
| 54 | 9 | 19 | 24 | 7 | 23 | 12 | 23 |
| 55 | 9 | 18 | 24 | 6 | 24 | 12 | 23 |
| 56 | 9 | 19 | 23 | 7 | 23 | 12 | 24 |
| 57 | 9 | 19 | 23 | 7 | 23 | 12 | 24 |
| 58 | 9 | 19 | 24 | 8 | 23 | 12 | 24 |
| 59 | 9 | 19 | 24 | 8 | 24 | 12 | 23 |
| 60 | 9 | 19 | 24 | 8 | 24 | 14 | 24 |
| 70 | 9 | 19 | 23 | 8 | 25 | 14 | 23 |
| Promedio | 13.5 | 19.5 | 23.5 | 11.5 | 22.5 | 16 | 23.5 |

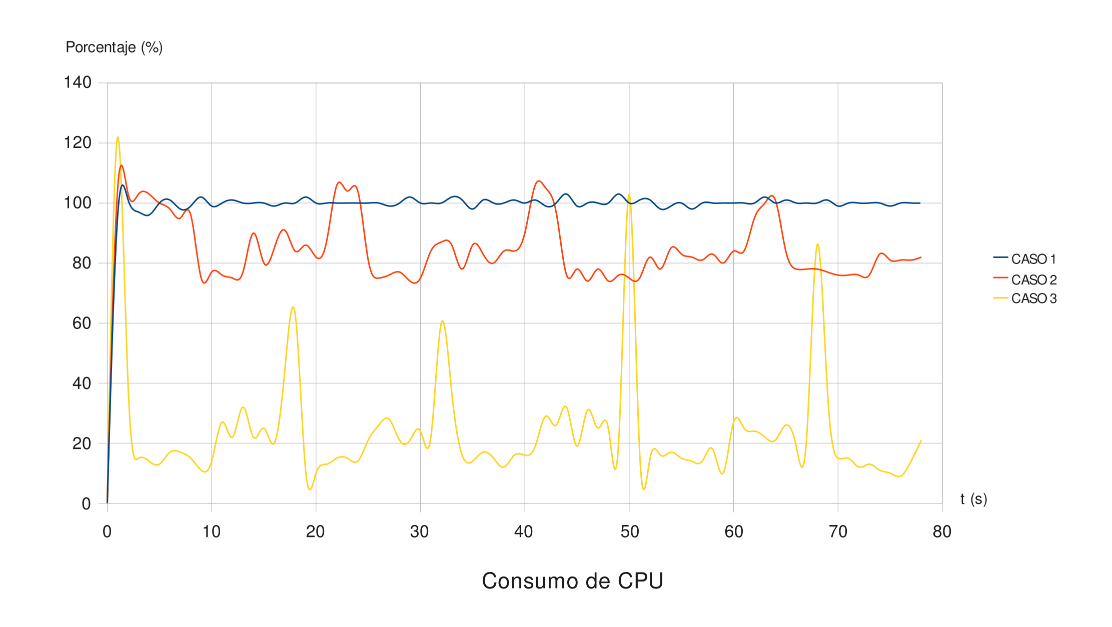



Figura 6.4.: Consumo de GPP en flujos de video para los casos 1,2,3

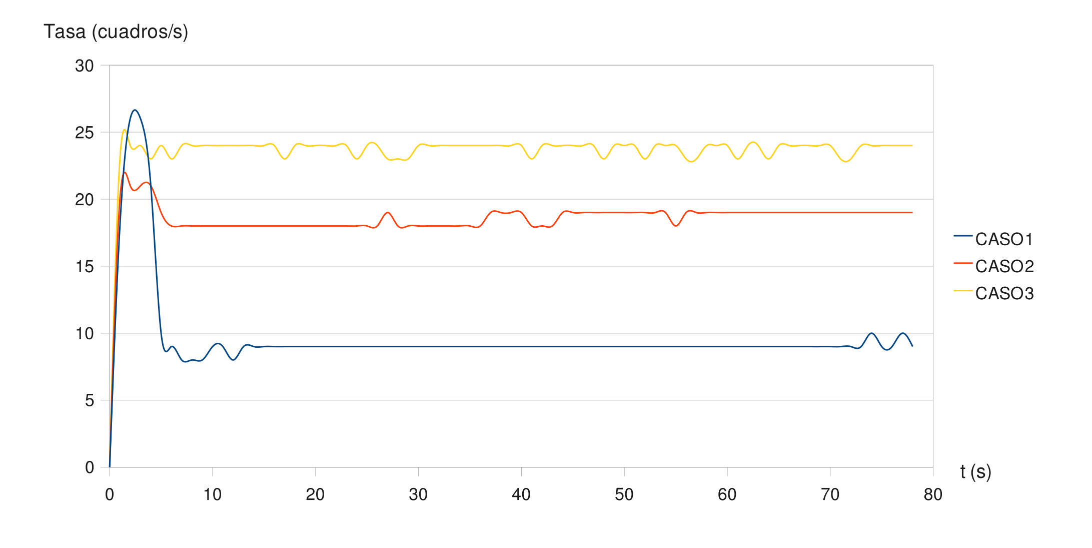



Figura 6.5.: Tasa de datos en el despliegue de video para los casos 1,2,3

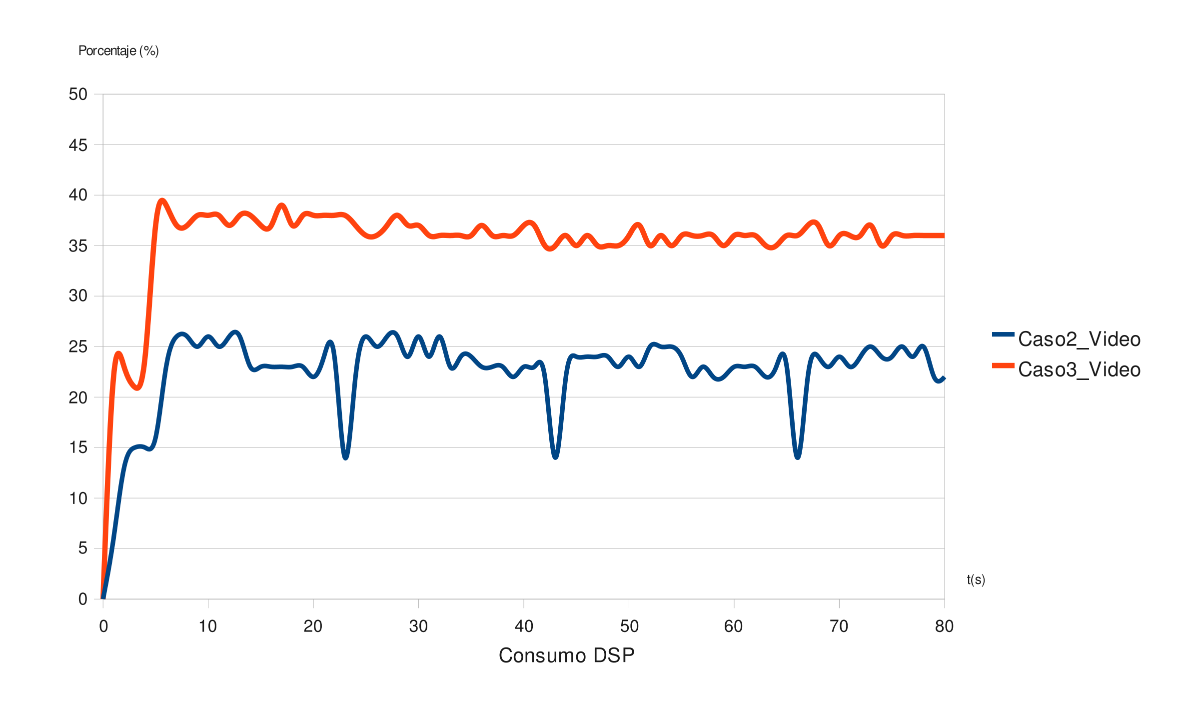



Figura 6.6.: Consumo de DSP para los casos 2 y 3 de video

Para el caso número 1 donde todos los procesos se encuentran en el GPP, el consumo del GPP tiene un promedio de 100 % para poder desplegar video, además se obtiene un promedio de tasa de imagénes por segundo de 13,5 cuadros/s. Añadiendo el elemento de decodificación en el DSP en el caso 2, se logra aumentar el rendmiento del GPP. En este caso se observa como los procesos en el GPP, alcanzan un promedio del 85 %. El procesador deja de estar saturado en el 100 % como en el caso anterior, liberando al procesador GPP para que pueda ejecutar otros procesos en paralelo al despliegue de video. También se puede observar cómo en este caso la tasa de imágenes por segundo aumentó a un promedio de 19,5 cuadros/s.

Por último se añadió el elemento de conversión de espacio de colores creado en este proyecto (*dt_colorspace*) ejecutado en el DSP. Las gráficas 6.4 y 6.5, muestran cómo para este caso, el consumo de GPP decae hasta un promedio de consumo de 24 %, y si se evalúa la tasa de cuadros por segundo ésta aumenta a un valor promedio de 23,5 fps, que significa una mejora de un 20.5 % con respecto al caso 2 y una mejora de un 74,5 % con respecto al caso 1 en este rubro.

En la figura 6.5 se puede observar para el caso 1 con respecto a la tasa de cuadros por segundo, que al inicio de ésta gráfica existe un pico máximo que luego se estabiliza. Este pico se debe al preprocesamiento de la aplicación que permite tener datos de video listos representados por búferes disponibles para ser desplegados una vez que inicia la aplicación. De esta forma la tasa de cuadros por segundo desplegada alcanza un máximo, para luego

estabilizarse aproximadamente después de 6 segundos, ya cuando otros servicios recargan al procesador ARM, disminuyendo así la capacidad para desplegar búferes a la salida.

En la figura 6.6 se observa el consumo de DSP para los casos 2 y 3. Es relevante observar en el caso 3 cuando se ejecutan dos módulos de algoritmos en el DSP, que el consumo no supera el 40 %, lo que resalta la capacidad del procesador DSP para este tipo de algoritmos matemáticos. La tabla 6.10 muestra un resumen de los datos tomados para los csos 1, 2 y 3.

Los casos 4 y 5 utilizan un archivo de video con formato h264 sin contenedor mp4. El caso 4 de estudio utiliza un elemento de decodificación de video para formatos h264 y un elemento de conversión de espacio de colores en el procesador GPP, y el caso 5 es la contraparte en el DSP. El rendimiento con respecto a la utilización de GPP y el despliegue de cuadros por segundo para estos casos se puede observar en la figuras 6.7 y 6.8 respectivamente, y la información detallada con respecto al flujo de datos de *GStreamer* utilizados se pueden observar en los apéndices A.3.2.4 y A.3.2.5.

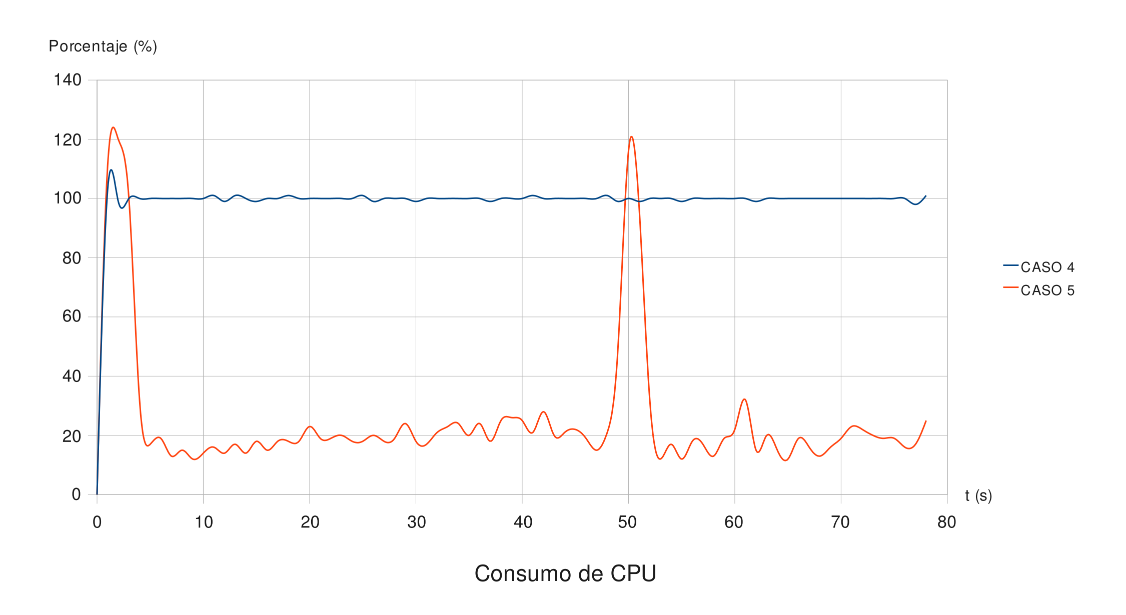



Figura 6.7.: Consumo de GPP en flujos de video para los casos 4 y 5

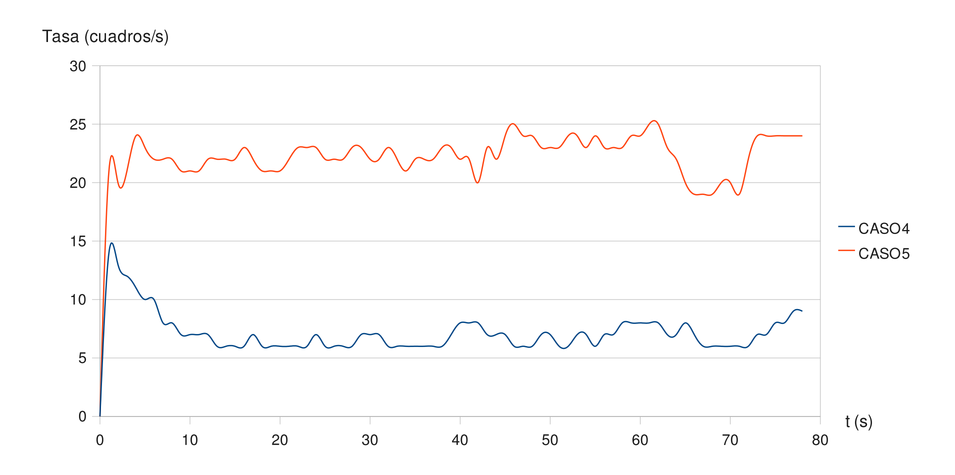



Figura 6.8.: Tasa de datos en el despliegue de video para los casos 4 y 5

Para estos casos, cuando se ejecuta la aplicación con sólo los elementos procesados en el GPP se utiliza una imágen de tamaño de $368 \times 246$ pixeles, mientras que para su contraparte se utiliza una imagen de tamaño $720 \times 480$ pixeles. Comparando rendimientos para ambos casos, se demuestra que una imagen de mayor tamaño no satura el procesador gracias a la integración de los procesos en el DSP, además la tasa de imágenes por segundo pasa de un promedio de 11,5 a 22,5 cuadros/s. Lo anterior demuestra, que además de mejorar el rendimiento de GPP, y la tasa de transferencia; la cantidad de datos de video a procesar también puede aumentar.

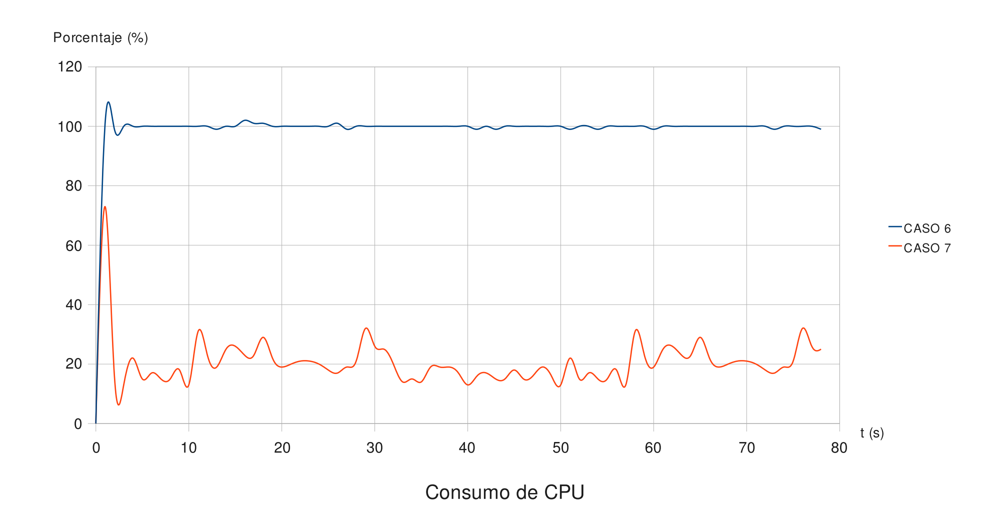



Figura 6.9.: Consumo de GPP en flujos de video para los casos 6 y 7

El caso 6 utiliza un elemento de decodificiación de audio en el procesador GPP y efecto de audio en el procesador GPP. El rendimiento con respecto a utilización de GPP se puede observar en la figura 6.10 y la información referente al flujo de GStreamer utilizado se puede obsevar en la sección A.3.2.6.

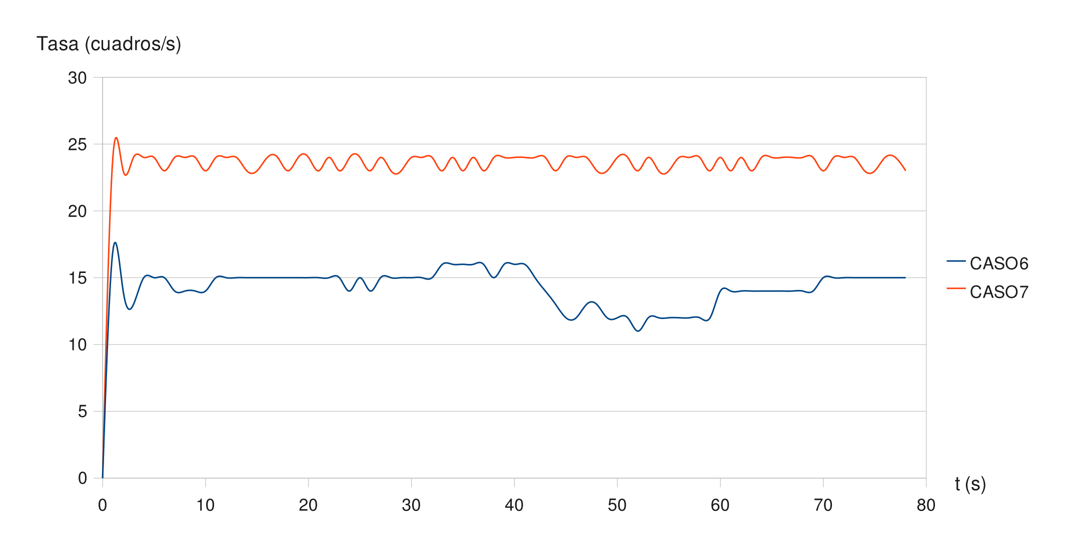



Figura 6.10.: Tasa de datos en el despliegue de video para los casos 6 y 7

Los casos 6 y 7 utilizan un archivo de video con formato mpeg4 sin contenedor mp4. El caso 6 de estudio utiliza un elemento de decodificación de video para formatos mpeg4 y un elemento de conversión de espacio de colores en el en el procesador GPP, y el caso 7 es la contraparte en el DSP. El rendimiento con respecto a la utilización de GPP y el despliegue de cuadros por segundo para estos casos se puede observar en la figuras 6.9 y 6.10 respectivamente, y la información detallada con respecto al flujo de datos de *GStreamer* utilizados se pueden observar en los apéndices A.3.2.6 y A.3.2.7. Su comportamiento en cuanto a la mejora de rendimientos es similar a los casos 4 y 5, donde el rendimiento de GPP y los cuadros por segundo aumentan cuando se utilizan los algoritmos de DSP.

## 6.3. Resumen

Una vez implementado el marco de trabajo propuesto para la creación de módulos de DSP con el estándar de *iUniversal,* junto a las herramientas que brinda Texas Instruments y las herramientas de trabajo integradas en el SDK para la creación, ejecución y manipulación dentro de la arquitectura OMAP-L138, para determinar el incremento en la puesta en marcha de un módulo de éstos se evaluaron tres casos de estudio.

El primero consistió en la realización misma del SDK, que parte de ninguna integración de herramientas y poca experiencia de los ingenieros. El proceso de creación del módulo tomó de 5 a 6 meses.

El segundo caso consiste en la puesta en marcha de un módulo ya usando el SDK, por parte de dos estudiantes del ITCR, que requirieron de 2 a 3 meses, para la implentación de un algoritmo filtro FIR en el DSP de esta arquitectura.

En el tercer caso se evaluó un proyecto de la empresa Ridgerun, el cual constaba de la creación de un algoritmo de audio en 3D ejecutándose en el DSP, implementado a partir de filtros FIR. En este caso, el ingeniero a cargo del proyecto se encontraba relacionado con la estructura de un SDK y logró implementar e integrar el algoritmo junto a una aplicación, con las herramientas provistas en el SDK en un tiempo de 2 a 3 semanas, lo que representa un 87.5 % en la mejora del tiempo con respecto al primer caso.

De estos tres casos se deduce que la documentación y el SDK permiten mejorar el tiempo de integración de un algoritmo módulo de DSP y las herramientas implementadas disminuyen la carga en la curva de aprendizaje considerablemente con respecto a la de este proyecto, tanto para personas no relacionadas con el ambiente del SDK, como para ingenieros de la empresa Ridgerun.

Resumiendo, los casos analizados tanto para audio como para video demuestran que la integración implementada con las herramientas del SDK para la creación de algoritmos que utilicen la arquitectura de DSP, permiten obtener mejoras en una arquitectura dual en los siguientes rubros: tiempo de implementación de algoritmos DSP, aplicaciones de prueba, rendimiento del GPP, y cantidad de datos a procesar, y además contienen herramientas que permiten medir el consumo de los procesadores (ARM y DSP), para observar los rendimientos de los algoritmos y las aplicaciones creadas.

# 7. Conclusiones y Recomendaciones

## 7.1. Conclusiones

Los tres casos de estudio analizados en la sección 6.3demuestran que el kit de desarrollo de software desarrollado permite pasar de 6 a 2 meses el tiempo de desarrollo de un módulo DSP compatible con la aquitectura OMAP-L138 para personas ajenas al SDK y para un ingeniero de la empresa el tiempo de desarrollo y puesta en marcha se disminuyó de 6 meses a 3 semanas.

La integración de herramientas de software permitió a la empresa RidgeRun obtener la primera versión estable de un kit de desarrollo de software empotrado para la arquitectura OMAP-L138 que incluye soporte para la implementación de módulos DSP con una estrutura basada en *iUniversal*.

La integración del paquete *DVSDK* en el kit de desarrollo de software pone a disposición una serie de herramientas que permiten medir el consumo de recursos de los procesadores GPP y DSP, cuando se ejecutan aplicaciones que utilizan módulos de algoritmos para DSP.

La implementación de un algoritmo de conversión de espacios de color en una aplicación de *GStreamer* demuestra la capacidad del SDK para integrar aplicaciones que involucran procesamiento de video en el sistema embebido *OMAP-L138 ZoomEvm*, a través del marco de trabajo de *Codec Engine* y *iUniversal*. Con esta implementación se logró:

- Mejorar la tasa de cuadros por segundo hasta en un 70 %.
- Mejorar el consumo de CPU hasta un 76 %.
- Aumentar hasta dos veces la cantidad de datos procesados en un video digital.
- Un consumo no mayor al 45 % del DSP.

La implementación de un algoritmo de filtro FIR en una aplicación de *GStreamer* demuestra la capacidad del SDK para integrar aplicaciones que involucran procesamiento de audio en el sistema embebido *OMAP-L138 ZoomEvm*, a través del marco de trabajo de *Codec Engine* y *iUniversal*. Con esta implementación se logró:

- Mejorar el consumo de CPU para formato AAC hasta un 81,12 %.
- Mejorar el consumo de CPU para formato MP3 hasta un 42,36 %.
- Un consumo no mayor al 14 % del DSP.

## 7.2. Recomendaciones

Existe una solución complementaria para una arquitectura que solo utilice el DSP, ya que la *ZoomEvm* permite cambiar el SoC que contiene DSP + ARM a uno que solo contiene un DSP. El marco de trabajo base se desarrollaría a través de un sistema operativo nativo para DSP, denominado *DSP/BIOS*. Este acercamiento conlleva el excluir el marco de trabajo de *CodecEngine* y enfocarse en la creación de los módulos DSP, igualmente compatibles con el estandar *XDAIS* a través de la *API* de *iUniversal*.

La implementación de este paquete permite el seguimiento y la depuración a nivel de hardware (memoria de procesos, datos, registros y otros componentes pertenecientes al *DSP*), de los programas creados que se ejecuten en este procesador, lo que permite una mayor interacción con la arquitectura del DSP.

Actualmente, con el marco de trabajo creado, sólo una aplicación en tiempo de ejecución es capaz de acceder al *CodecServer*.

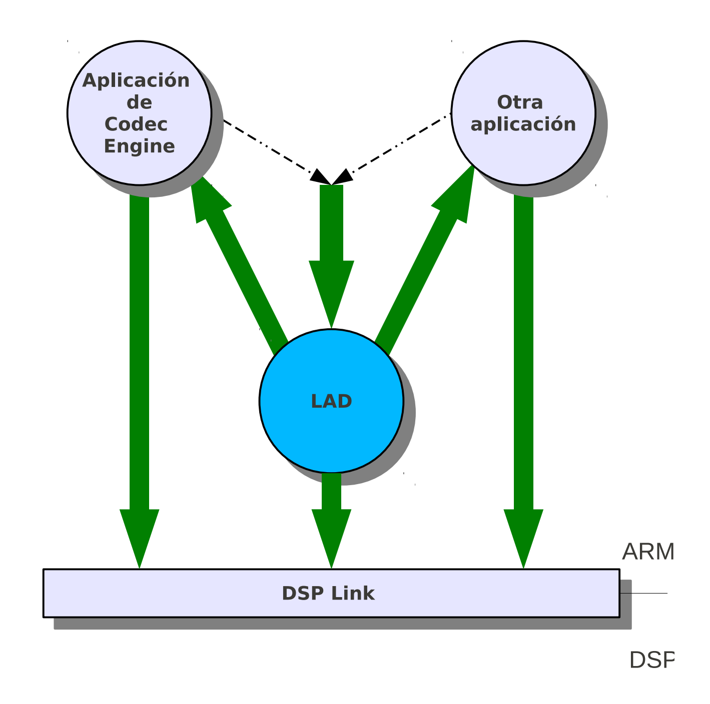



Figura 7.1.: Esquema de LAD dentro del marco de trabajo creado

Implementar las funciones de *LAD* (*Link Arbitrer Daemon*), en futuras versiones dentro del paquete de *CodecEngine,* permitiría ejecutar más de una instancia del mismo *CodecServer* (hilos en el DSP). Con la integración de LAD se pueden crear *n* aplicaciones, sin usar necesariamente el marco de *CodecEngine* (con un límite máximo) que pueden crear a su vez *n* instancias de un módulo algoritmo de DSP. La figura 7.1 ilustra del proceso que ejecutaría LAD.

# Bibliografía


[1] A. Doblander, A. Zoufal and B. Rinner, *“A novel software framework for embedded multiprocessor smart cameras*”, ACM Trans. Embed. Comput. Syst., vol. 8, no. 3, pp. 1– 30, 2009.

[2] A. Ford and A. Roberts, *Colour Space Conversions*, (b) ed., August 1998.

[3] A. Lad, P. Raghavan and S. Neelakandan, *Embedded Linux System Design and Development.* Taylor & Francis Inc, 2006.

[4] C. Almeida, “*Practical experience teaching embedded systems*”, SIGBED Rev., vol. 5, no. 3, pp. 1–8, 2008.

[5] D. A., Kerr, *”Chrominance Subsampling in Digital Images*”, 2 edition, (December 2009) [Online]. Available: http://dougkerr.net/Pumpkin/articles/Subsampling.pdf. [Accessed: January 11, 2010]

[6] D. J. Jackson and P. Caspi, “*Embedded systems education: future directions, initiatives, and cooperation*”, SIGBED Rev., vol. 2, no. 4, pp. 1–4, 2005.

[7] D. P. Bovet and M. C. Ph, *Understanding the Linux Kernel, Third Edition*. O’Reilly Media, 3 ed., November 2005.

[8] Texas Instruments, *“DaVinci-PSP-SDK Product Downloads*”, DaVinci_03_20 Product Download Page, (March 2010) [Online]. Available: http://software-dl.ti.com/dsps/dsps_public_sw/sdo_tii/psp/LinuxPSP/DaVinci_03_20/index.html, [Accessed: January 10, 2010].

[9] G. D. Micheli, W. Wolf, and R. Ernst, *Readings in Hardware/Software Co-Design*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001.

[10] I. E. G. Richardson, *Video Codec Design: Developing Image and Video Compression Systems*. Wiley, 2002.

[11] J. Lombardo, *Embedded Linux*, Thousand Oaks, CA, USA: New Riders Publishing, 2001.

[12] J.G. Proakis and D.K. Manolakis, *Digital Signal Processing* (4th Edition) Prentice Hall, Hardcover, 2006

[13] K. Hilman, “*Kernel development tree for ti davinci family of processors*”.(2009). [Online]. Available: http://git.kernel.org/?p=linux/kernel/git/khilman/linux-davinci.git;a=summary [Accessed: December 15, 2009]

[14] Logic Corporate,“ *OMAP-L138 Zoom Evm development kit* ”, Innovate product solutions (July, 29 2009). [Online]. Available: http://www.logicpd.com/products/development-kits/zoom-omap-l138-evm-development-kit. [Accessed: December 15, 2009]

[15] M. Barr and A. Massa, *Programming Embedded Systems: With C and GNU Development Tools*. O'Reilly Media, Inc., 2006.

[16] M. Bramberger, J. Brunner, and B. Rinner, "*Real-time video analysis on an embedded smart camera for traffic surveillance*", in RTAS '04: Proceedings of the 10th IEEE Real- Time and Embedded Technology and Applications Symposium, (Washington, DC, USA), p. 174, IEEE Computer Society, 2004.

[17] M. Jan, "*Digital Audio Technology: A Guide to CD, MiniDisc, SACD, DVD(A), MP3 and DAT; electronic version*" , Elsevier, San Diego, CA, 2001.

[18] M. Kahrs and K. Brandenburg, *Applications of digital signal processing to audio and acoustics*. Norwell, MA, USA: Kluwer Academic Publishers, 1998.

[19] M. Kaufmann, *Digital Video and HDTV Algorithms and Interfaces*. San Francisco, CA, USA: Poynton C. Publishers Inc., 2003.

[20] M. Wang and X. Fu, *IS-127 Enhanced Variable Rate Speech Coder: Multichannel TMS320C62x Implementation, C6000 Applications*. Texas Instruments, Literature Number: SPRA566A , June 1999.

[21] A. Author, "Document title," Webpage name, Source/production information, Date of internet publication. [Format]. Available: internet address. [Accessed: Date of access].

[22] O. S. Community, "*Gstreamer: open source multimedia framework*", (2005).[Online]. Available: http://www.gstreamer.net/. [Accessed: January 12, 2010].

[23] R. M. Stallman, Richard M. and P. D. Smith., *GNU Make Manual - GNU Project*. Free Software Foundation (FSF), 3.81 ed., April 2006.

[24] RR. I. Community, "*SDK FAQ. - RidgeRun Developer Connection*," (February 2010). [Online]. Available: https://www.ridgerun.com/developer/wiki/index.php/SDK_F.A.Q.. [Accessed: May 15, 2010].

[25] RR. I. Community, "*RidgeRun embedded solutions home page*," (February 2010). [Online]. Available: http://www.ridgerun.com. [Accessed: May 25, 2010].

[26] RR. I. Community, "*Software developer kit user's guide*," February 2006.

[27] J C. Russ, *Image Processing Handbook*, Fourth Edition, CRC Press, Inc., 2002

[28] "Sourcery g++ lite 2009q1-203 for ARM GNU/Linux," (1997). [Online]. Available: http://www.codesourcery.com/sgpp/lite/arm/portal/release858. [Accessed: January 10, 2010].

[29] Texas Instruments, "*Davinci psp software development kit (sdk) 03.20 updates*", May 2010. v03.20.00.08. [Online]. Available: http://software-dl.ti.com/dsps/dsps_public_sw/sdo_tii/psp/LinuxPSP/DaVinci_03_20/03_20_00_08/index_FDS.html. [Accessed: May 22, 2010].

[30] Texas Instruments, "*Ti gstreamer dmai plug-in for davinci and omap processors - gstreamer - ti software folder,*" (November 2009). [Online]. Available: http://focus.ti.com/docs/toolsw/folders/print/gstreamer.html. [Accessed: December 2, 2010].

[31] Texas Instruments, *How to Create Delay-based Audio Effects on the TMS320C672x DSP*, Literature Number: SPRAA5, November 2005.

[32] Texas Instruments, *Making DSP Algorithms Compliant with the eX-pressDSP*, Literature Number: SPRA579A, January 2000.

[33] Texas Instruments, *TMS320C6000. DSP/BIOS. User s Guide*, Literature Number: SPRU303B, March 2000.

[34] Texas Instuments, *xDAIS-DM Digital Media User Guide*, Literature Number: SPRUEC8B, January 2007.

[35] Texas Instruments, *XDC Consumer User Guide*, Literature Number: SPRUEX4, July 2007.

[36] Texas Instruments, *Codec Engine Application Developer User's Guide*, Literature Number: SPRUE67D, September 2009.

[37] Texas Instruments, *TMS320C64x+ DSP Image/Video Processing Library (v2.0.1)*, Literature Number: SPRUF30A, October 2007. Revised May 2008.

[38] Texas Instruments, *AM17x/AM18x ARM Microprocessor Liquid Crystal Display Controller (LCDC) User Guide*, Literature Number: SPRUFV5, March 2010.

[39] Texas Instruments, *AM17x/AM18x ARM Microprocessor Multichannel Audio Serial Port (McASP)*, Literature Number: SPRUFV6, March 2010

[40] Texas Instruments, *TMS320C6748 Fixed/Floating Point Digital Signal Processor*, Literature Number: SPRUGJ7D, August 2009.

[41] Texas Instruments, *TMS320 DSP Algorithm Standard Rules and Guidelines*, Literature Number: SPRU352G, June 2005–Revised February 2007.

[42] Texas Instruments, *TMS320 DSP Algorithm Standard API Reference,* Literature Number: SPRU360E, February 2005. Revised Febrero 2007.

[43] Texas Instruments, *TMS320 DSP/BIOS v5.41 User's Guide*, Literature Number: SPRU423H, August 2009.

[44] Texas Instruments, *TMS320C6000 Optimizing C Compiler Tutorial*, Literature Number: SPRU425A , August 2002.

[45] S. J. Sangwine and R. E. N. Horne, *The Colour Image Processing Handbook (Optoelectronics, Imaging and Sensing)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc, 1998.

[46] S. Preissig, *Programming Details of Codec Engine for DaVinci TM Technology*. Texas Instruments, Literature Number: SPRY091, November 2006.

[47] Texas Instruments, *OMAP-L138 Low-Power Applications Processor*, Literature Number: SPRS586A, June 2009. Revised August 2009.

[48] W. Denk . "*Das u-boot source tree*". [Online]. Available: http://git.denx.de/?p=u-boot.git;a=summary. [Accessed: January 5, 2010 ].

[49] X. Amatriain and U. Zölzer , *DAFX - Digital Audio Effects*. John Wiley & Sons, 2002.

[50] U. Zölzer: *Digital audio signal processing*, John Wiley & Sons, 1997

# A. Apendices

## A.1. Manual de usuario

# How to Create DSP Algorithm Package with RidgeRun SDK

Esteban Zúñiga Mora

esteban.zuniga@ridgerun.com

Jorge Hidalgo Chaves

jorge.hidalgo@ridgerun.com

### Introduction

This document will be an easy way to create a new DSP algorithm based on TMS320C6748 DSP processor. It will target these algorithms and assists the reader with making the algorithms compliant with the TMS320 DSP algorithm standard which is part of TI's eXpressDSP technology initiative. The document proposes a simple approach to running xDAIS (iUniversal) algorithms with a Codec Engine (CE) application. The wizard selected is responsible for creating an xDAIS algorithm (iUniversal), and providing the necessary packaging to enable these algorithms to be consumed and configured by Codec Engine.

---



We assume readers have a basic understanding of the elements in a Codec Engine application, including VISA APIs, xDM, servers, and codecs.

## References

For details, see the following documents:

- TMS320 DSP Algorithm Standard Rules and Guidelines ***SPRU352E***
- TMS320 DSP Algorithm Standard API Reference ***SPRU360E***
- TMS320 DSP Algorithm Standard Developer's Guide ***SPRU424C***
- Codec Engine Algorithm Creator User's Guide ***SPRUED6C***
- Based on the Codec Engine GenCodecPkg ***Wizard GenCodecPkg***
- The most important care-abouts in a typical Codec Engine configuration ***Codec Engine configuration en breve***

## Requirements for the GencodecPkg Wizard

- Engine 2.25 or later
  - CE 2.25.02
- improves the GUI
- interface XDAIS 6.25or later
- XDAIS 6.25.02 improves the code templates
- XDCtools 3.16 or later

## Making a basic package xDAIS & xDM compliant

Those steps shown above has to be followed to complete a memcpy DSP algorithm. If you desire to add new features , new functions, and others refer to previous section links for more details.

### Adding the wizard support to DVSDK makefile

The GenCodecPkg wizard generates the files and packaging required for integrating an algorithm into Codec Engine. Assuming your using a RR SDK you have a complete integration with a TI DVSDK release, then you likely have already defined paths to all your tools in the Rules.make.

Those generated Codec Packages can be integrated into a Server and the document will explain the way to append the new codec into TI's Codec Server.

### Checking your DVSKD Makefile Support

```
$ cd $(DEVDIR)/propietary/dvsdk*
$ gedit Makefile
```

On the Makefile check if the gencodecpkg macro is activated if it's already added, please ignore next step.

### Inside the Makefile from the DVSDK

```
$ cd $(DEVDIR)/propietary/dvsdk*
$ gedit Makefile
```

Append the following rule:

```
gencodecpkg:
	$(XDC)/xs --xdcpath="$(CE_INSTALL_DIR)/packages; \
	$(XDAIS_INSTALL_DIR)/packages" ti.sdo.ce.wizards.gencodecpkg
```

Make sure you use a **tab** instead **spaces**.

### Running the wizard

From your DVSDK directory you can invoke gencodecpkg with the following command:

```
$ cd $(DEVDIR)/proprietary/dvsdk_*
$ make gencodecpkg
```

### At the First Screen

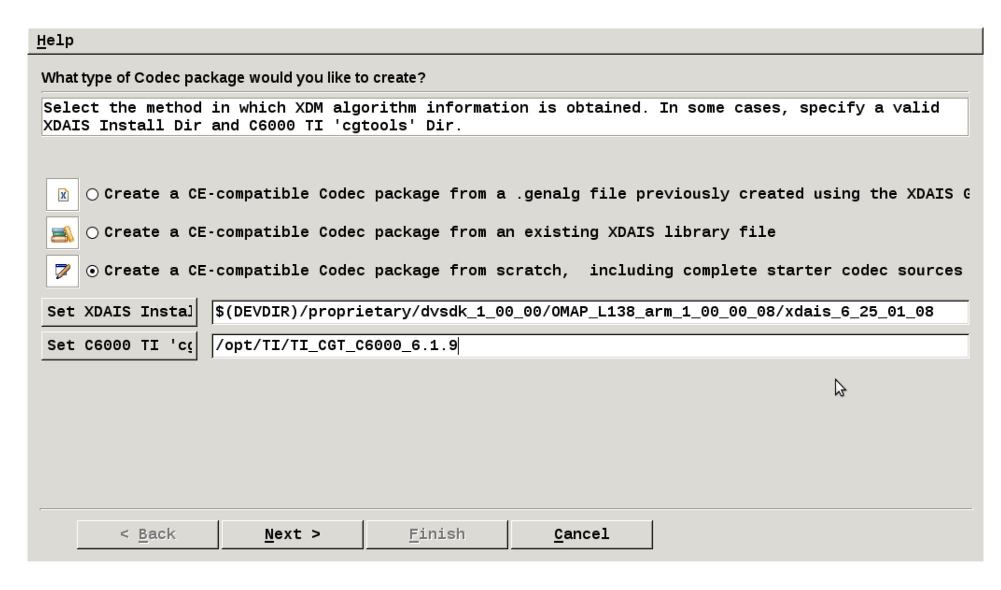



Figura A.1.: First Window Selection for the GenCodePkg Wizard

1. Choose the 3rd option, **I want to create an algorithm from scratch**

2. Point to the xDAIS directory Point to the root of the compiler installation: **CGTOOLS**
3. Click on next

### At the Second Screen

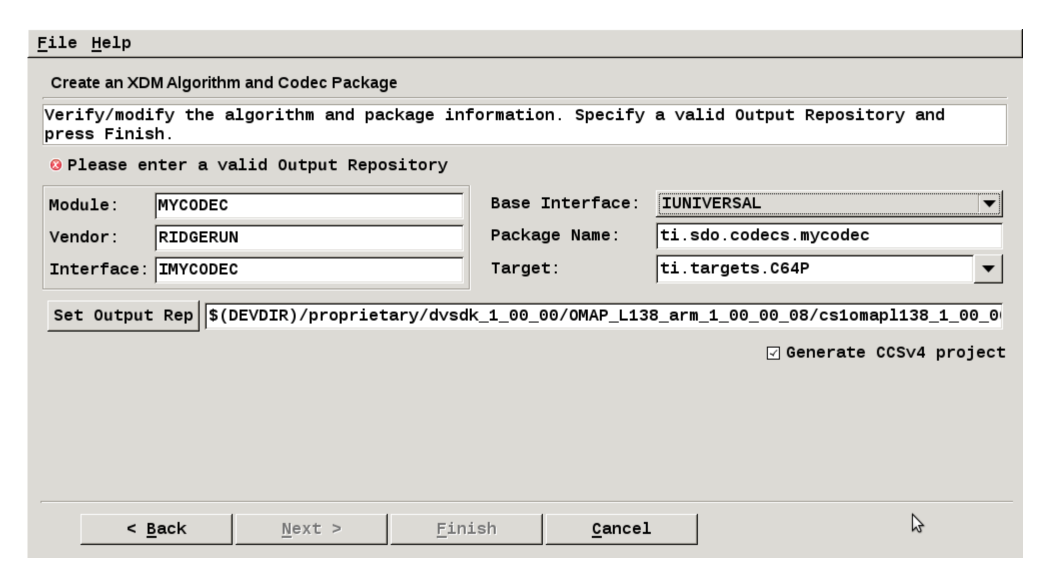



Figura A.2.: xDM and Codec Package Wizard

- Fill the spaces according with the name of your new package.
- Keep the capital letters on the names. as it's shown on the table placed below

| Space | Examples |
|---|---|
| Module | MYCODEC |
| Vendor | RIDGERUN |
| Interface | IMYCODEC |
| Package name | ti.sdo.codecs.mycodec |
| Base Interface | IUNIVERSAL. |
| Target | ti.targets.C64P |
| Output Dir | $(DVSDK)/csomapl138/packages |

- Click Finish to generate the starter files.

## Create the CE-Consumable Codec Package

After creating the xDM algorithm and Codec Package Structure, made an extra package called ti.sdo.codecs.codecs.*.ce (**where * is the codec eg: mycodec**) that declares static properties relevant to the Codec Engine.

- The file **MYCODEX.xdc** declares the xDM class we are trying to implement and the algorithm function table.
- The file **MYCODEC.xs** provides the getStackSize() function, which is used when building an application for the Codec Engine to declare the combined amount of stack usage for the algorithm.
- Inside the ti.sdo.codecs.codecs.mycodec.ce directory create the files **package.xdc** and **package.bld.**

### Creating ce directory

Go to the codec path and create the ce directory

```
$ cd (...) cs1omap/packages/ti/sdo/codecs/mycodec
$ mkdir ce
```

Move the MODULE.xdc and the MODULE.xs to "ce" directory

```
$ mv MYCODEC.xdc ce/
$ mv MYCODEC.xs ce/ [edit]
```

### Modifing the files

#### MODULE.xdc

Modify the file MODULE.xdc and add the code highlighted as shown below:

```
metaonly module MYCODEX
inherits ti.sdo.ce.universal.IUNIVERSAL
{
     readonly config ti.sdo.codecs.mycodec.MYCODEC.Module alg =
ti.sdo.codecs.mycodec.MYCODEC;
      override readonly config String ialgFxns = "MYCODEC_TI_MYCODEC";
}
```

#### package.xdc

It declares the name of the package and the module it contains. Furthermore, it uses the “requires” statement to specify its dependency on the algorithm and adapter package. Hence when the server package is configured to use this CE-consumable package, it picks up the correct libraries from the packages it requires.

The **package.xdc** file is the package definition file, which defines your Codec Server’s name and its dependencies. The package name must reflect the directory structure under the ti codecs.

```
requires ti.sdo.ce.universal;
requires ti.sdo.codecs.mycodec;
/* ======== package.xdc ========
* Provides MYCODEC interface adapter for
* ti.sdo.codecs.mycodec codec.
*/
package ti.sdo.codecs.mycodec.ce [1, 0, 0]
{
   module MYCODEC;
}
```

#### package.bld

```
/* ======== package.bld ======== */
Pkg.attrs.exportAll = true;
```

## Root Codec Directory

#### MYCODEC.xdc

At the root directory of the codec , create a file named MYCODEC.xdc

```
/* ======== MYCODEC========*/
metaonly module MYCODEC {
/* ======== watermark ======== */
config Bool watermark = false;
/* ======== Code Section ======== */
config String codeSection="DDR2";
/* ======== Uninitialized Data Section ======== */
config String udataSection="DDR2";
/* ======== Initialized Data Section ======== */
config String dataSection="DDR2";
}
```

#### package.bld

File located at the root codec directory where the sources of your algorithm has to be added. Describes the steps required to build and release the package. Contains JavaScript instructions for building any libraries, object files, or executables that are part of the package, and specifications of what goes into a release package.

Additionally, any source and header files needed to build the artifacts and any pre-built files to be included in a release package must be included in the package directory. The source or header files used in the package build process are not included in a release package, unless you explicitly add them using if you desire to add new sources modify the code eg :

```
var SRCS = ["mycodec.c", "algorithm.c"];
```

Where mycodec. is the IUNIVERSAL interface and the algorihtm.c is the where you DSP algo will be located. For example, by adding the following line to package.bld, all files in the src and include directories and the file readme.txt are released:

```
Pkg.otherFiles = [ 'src', /* the whole src subdirectory */
                   'include', /* the whole include subdirectory */
                   'readme.txt', ];
```

In order to include a library, legacy.lib, which was built outside the package using a legacy build system, you can copy the library into the package directory and include it in Pkg.otherFiles in the package.bld file.

```
Pkg.otherFiles = [ 'legacy.lib', 'readme.txt', ];
```

#### package.xs

Package properties that can vary across platforms and configurations. The getLibs() function returns the library name that the package exports. The XDC build system calls this special function—whenever an XDC application "uses" a package—to link against the package library matching the program's configuration. The exported library does not have to be built using the housing package's build script. It can be built with any legacy build system and placed into the package.

The string "64P" is a code for DSP target binaries. (ARM target binaries have a different code, "MVArm9".) If you are building an ARM application and this codec package gets consumed, the condition above is false and the getLibs() return an empty library name—which is what you want when you don't provide any libraries for the ARM.

In the file package.xs at the root directory of the codec append the code shown below (This code is used by the linker):

```
/* * ======== getSects ======== */
function getSects()
{
var template = null;
     if (Program.build.target.isa == "64P")
     {
        template = "ti/sdo/codecs/audio_effect/link.xdt";
      }
return (template);
}
```

#### link.xdt

Create a file named link.xdt at the root directory of codec and modify this file, Change MYCODEC by your codec name Change VENDOR by your vendor name.

```
SECTIONS
{
% if (this.MYCODEC.dataSection)
{ .const:.string > ‘this.AUDIO_EFFECT.dataSection‘
%}
% if (this.MYCODEC.codeSection)
{
.text:_algorithm > ‘this.MYCODEC.codeSection‘
.text:algProcess:_MYCODEC_VENDOR_process > ‘this.MYCODEC.codeSection‘
.text:algInit:_MYCODEC_VENDOR_initObj > ‘this.MYCODEC.codeSection‘
.text:algAlloc:_MYCODEC_VENDOR_alloc > ‘this.AMYCODEC.codeSection‘
.text:algFree:_MYCODEC_VENDOR_free > ‘this.MYCODEC.codeSection‘
.text:algControl:_MYCODEC_VENDOR_control > ‘this.MYCODEC.codeSection‘
%}
}
```

Usually the codec should be integrated with the codec server that comes with the dvsdk out-of-the-box.

## Codec server integration

There are two options here, one of them is to add your codec to the current codec server provided by the dvsdk or you can create a new codec server. Adding the codec to the current codec server provided by the dvsdk:

### Add the codec

Modify the file at

For adding your codec as follows:

```
var MYCODEC = xdc.useModule('ti.sdo.codecs.mycodec.ce.MYCODEC');
// Module Config
MYCODEC.alg.watermark = false;
MYCODEC.alg.codeSection = "DDR2";
MYCODEC.alg.dataSection = "DDR2";
MYCODEC.alg.udataSection = "DDR2";
```

### Array of algorithms

Then you should add your algorithm to the array of algorithms:

```
Server.algs = [
...
{name: "mycodec", mod: MYCODEC, threadAttrs:
{ stackMemId: 0, priority: Server.MINPRI + 1},
groupId : 2, }, ];
```

### Validate the codec

Add you codec to the validate function at file

`$(DEVDIR)/proprietary/<DVSDK>/<CODEC_SERVER>/packages/ti/sdo/server/cs/package.xs`

Validate your new codec with the code shown below:

```
function validate() {
...
validate_one_codec( "ti.sdo.codecs.mycodec", "MYCODEC" );
}
```

### Compile the codec server

## A.2. Ridgerun

***RidgeRun - Embedded Solutions*** *specialize in Embedded Software. We have Linux software development kits (SDKs), customized Linux kernels and drivers, complete reference applications and engineering services.*
*RidgeRun is a full solution Linux software and services business. We are a complete provider for your embedded Linux needs. Starting a new product development effort? Our team can provide a customized operating environment (small foot print, latest Linux Kernel) and drivers for your hardware platform. We also sell pre-packaged SDK's to help your team build network based, media based or wireless solutions. Need help with getting your software done? RidgeRun provides excellent software engineering services at near-shore competitive rates.*

#### Información de la Empresa

**Nombre:** RidgeRun - Embedded Solutions **Zona:** Cartago, La Union , San Juan

**Dirección exacta :** Del Hipermas de Curridabat 400 mts este , carretera a Tres Rios , Centro Comercial PLaza Magnolia

**Teléfono:** 22 71 14 87 **Fax:** — **Apartado:**—-

**Actividad Principal:** Diseño y desarrollo de sistemas embebidos.

#### Asesor en la empresa

**Nombre:** Diego Dompe Gamboa.

**Puesto que ocupa:** Embedded Software Team Lead.

**Departamento**: SDK Team.

**Profesión:** Ing en software. **Grado académico:** Bachiller.

**Teléfono:** 88 66 54 45. **Email:** diego.dompe@ridgerun.com.

## A.3. Casos de estudio

Módulo de CMEM utilizado para las pruebas de rendimiento de Audio y Video :

```
modprobe cmemk phys_start=0xC2200000 phys_end=0xC3200000
pools=1x5250000,12x 806400,10x153600
```

### A.3.1. Flujo de datos de Gstreamer de audio para medir consumo de CPU

#### A.3.1.1. Caso 1 de audio

| | |
|---|---|
| *Decodificador* | *faad (ARM)* |
| *Efecto* | *audioecho (ARM)* |
| *Formato audio:* | *acc* |
| *Contenedor de audio* | - |

*Flujo de datos de Gstreamer utilizado:*

```
gst-launch filesrc location= davincieffect.aac ! faad ! audioconvert ! audioecho
delay=500000000 intensity=0.6 feedback=0.4 ! audioconvert ! alsasink
```

#### A.3.1.2. Caso 2 de audio

| | |
|---|---|
| *Decodificador* | *TIAuddec1 (DSP)* |
| *Efecto* | *audioecho (ARM)* |
| *Formato audio:* | *acc* |
| *Contenedor de audio* | - |

*Flujo de datos de Gstreamer utilizado:*

```
gst-launch filesrc location= davincieffect.aac ! typefind ! TIAuddec1 ! audioconvert !
audioecho delay=500000000 intensity=0.6 feedback=0.4 ! audioconvert ! alsasink
```

#### A.3.1.3. Caso 3 de audio

| | |
|---|---|
| *Decodificador* | *faad (ARM)* |
| *Efecto* | *dt_audioeffect* (DSP) |
| *Formato audio:* | *acc* |
| *Contenedor de audio* | - |

*Flujo de datos de Gstreamer utilizado:*

```
gst-launch filesrc location= davincieffect.aac ! faad ! dt_audioeffect g1=4 g2=4 g3=4 g4=4 !
audioconvert ! alsasink
```

#### A.3.1.4. Caso 4 de audio

| | |
|---|---|
| *Decodificador* | *TIAuddec1 (DSP)* |
| *Efecto* | *dt_audioeffect* (DSP) |
| *Formato audio:* | *acc* |
| *Contenedor de audio* | - |

*Flujo de datos de Gstreamer utilizado:*

```
gst-launch filesrc location= davincieffect.aac ! typefind ! TIAuddec1 ! dt_audioeffect g1=4
g2=4 g3=4 g4=4 ! audioconvert ! alsasink
```

#### A.3.1.5. Caso 5 de audio

| | |
|---|---|
| *Decodificador* | *mad (ARM)* |
| *Efecto* | *audioecho* (ARM) |
| *Formato audio:* | *mp3* |
| *Contenedor de audio* | - |

*Flujo de datos de Gstreamer utilizado:*

```
gst-launch filesrc location= davincieffect.mp3 ! mad ! audioconvert ! audioecho
delay=500000000 intensity=0.6 feedback=0.4 ! audioconvert ! alsasink
```

#### A.3.1.6. Caso 6 de audio

| | |
|---|---|
| *Decodificador* | *mad (ARM)* |
| *Efecto* | *dt_audioeffect* (DSP) |
| *Formato audio:* | *mp3* |
| *Contenedor de audio* | - |

*Flujo de datos de Gstreamer utilizado:*

```
gst-launch filesrc location= davincieffect.mp3 ! mad ! audioconvert ! dt_audioeffect g1=4
g2=4 g3=4 g4=4 ! audioconvert ! alsasink
```

## A.3.2. Flujo de datos de Gstreamer de video para medir consumo de CPU

### A.3.2.1. Caso 1 de video

| | |
|---|---|
| *Decodificador* | *ffdec_h264 (ARM)* |
| *Conversión de Espacio de colores* | *ffmpegcolorspace* (ARM) |
| *Formato de video:* | *h264* |
| *Contenedor de video* | *mp4* |
| *Resolución:* | *320x240 pixels* |
| *Cuadros por segundos alcanzados* | *8 a 10 fps* |

*Flujo de datos de Gstreamer utilizado:*

```
gst-launch filesrc location= ShowoffIntro_h264.mp4 ! qtdemux ! ffdec_h264 !
ffmpegcolorspace ! fbdevsink
```

### A.3.2.2. Caso 2 de video

| | |
|---|---|
| *Decodificador* | *TIViddec2 (DSP)* |
| *Conversión de Espacio de colores* | *ffmpegcolorspace* (ARM) |
| *Formato de video:* | *h264* |
| *Contenedor de video* | *mp4* |
| *Resolución:* | *320x240 pixels* |
| *Cuadros por segundos alcanzados* | *17 a 18 fps* |

*Flujo de datos de Gstreamer utilizado:*

```
gst-launch filesrc location= ShowoffIntro_h264.mp4 ! qtdemux ! TIViddec2 ! ffmpegcolorspace
! fbdevsink
```

### A.3.2.3. Caso 3 de video

| | |
|---|---|
| *Decodificador* | *TIViddec2 (DSP)* |
| *Conversión de Espacio de colores* | *dt_colorspace* (DSP) |
| *Formato de video:* | *h264* |
| *Contenedor de video* | *mp4* |
| *Resolución:* | *320x240 pixels* |
| *Cuadros por segundos alcanzados* | *24 a 25 fps* |

*Flujo de datos de Gstreamer utilizado:*

```
gst-launch filesrc location= ShowoffIntro_h264.mp4 !  qtdemux !  TIViddec2 !  dt_colorspace !
fbdevsink
```

#### A.3.2.4. Caso 4 de video

| | |
|---|---|
| *Decodificador* | *ffdec_h264 (ARM)* |
| *Conversión de Espacio de colores* | *ffmpegcolorspace* (ARM) |
| *Formato de video:* | *h264* |
| *Resolución:* | *368x246 pixels* |
| *Cuadros por segundos alcanzados* | *6 a 8 fps* |

*Flujo de datos de Gstreamer utilizado:*

```
gst-launch filesrc location= davinci_cif.h264 !  typefind !  ffdec_h264 !  ffmpegcolorspace !
fbdevsink
```

#### A.3.2.5. Caso 5 de video

| | |
|---|---|
| *Decodificador* | *TIViddec2 (DSP)* |
| *Conversión de Espacio de colores* | *dt_colorspace* (DSP) |
| *Formato de video:* | *h264* |
| *Resolución:* | *720x480 pixels* |
| *Cuadros por segundos alcanzados* | *17 a 20 fps* |

*Flujo de datos de Gstreamer utilizado:*

```
gst-launch filesrc location= davincieffect_ntsc.264 !  typefind !  TIViddec2 !  dt_colorspace
!  fbdevsink
```

#### A.3.2.6. Caso 6 de video

| | |
|---|---|
| *Decodificador* | *ffdec_mpeg4 (ARM)* |
| *Conversión de Espacio de colores* | *ffmpegcolorspace* (ARM) |
| *Formato de video:* | *mpeg4* |
| *Contenedor de video* | *mp4* |
| *Resolución:* | *512x288 pixels* |
| *Cuadros por segundos alcanzados* | *13 a 14 fps* |

*Flujo de datos de Gstreamer utilizado:*

```
gst-launch filesrc location= prueba.mp4 !  qtdemux !  ffdec_mpeg4 !  ffmpegcolorspace !
fbdevsink
```

#### A.3.2.7. Caso 7 de video

| | |
|---|---|
| *Decodificador* | *TIViddec2 (DSP)* |
| *Conversión de Espacio de colores* | *dt_colospace* (ARM) |
| *Formato de video:* | *mpeg4* |
| *Contenedor de video* | *mp4* |
| *Resolución:* | *512x288 pixels* |
| *h2Cuadros por segundos alcanzados* | *24 a 25 fps* |

*Flujo de datos de Gstreamer utilizado:*

```
gst-launch filesrc location= prueba.mp4 !  qtdemux !  TIViddec2 !  dt_colorspace !  fbdevsink
```

## A.4. Licencia de la biblioteca Imglib

Software License Agreement

Important – Read carefully. If you do not agree with the following terms you may not use the software programs or associated documentation for any purpose: This Software License Agreement (or "Agreement") is a legal agreement between you (either an individual or entity) and Texas Instruments Incorporated ("TI"). The "Licensed Materials" subject to this Agreement include the downloadable software programs and documentation that accompany this Agreement and any "on-line" or electronic documentation associated with the software programs. Any applications included with the "Licensed Materials" are specifically designed and licensed for execution solely and exclusively on semiconductor devices manufactured by or for TI. By installing, copying or otherwise using the Licensed Materials, you agree to abide by the terms of this Agreement.

This Agreement is displayed for you to read prior to downloading and using the Licensed Materials. If you choose not to agree with these provisions, do not download or install the Licensed Materials.

1. **Intellectual Property, Title and Limited License Grant –** The Licensed Materials are protected by copyright laws, international copyright treaties, and trade secret laws, as well as other intellectual property laws and treaties. The Licensed Materials are licensed, not sold to you, and can only be used in accordance with the terms of this Agreement. TI retains title and ownership of the Licensed Materials, including all intellectual property rights in the Licensed Materials and all copies of the Licensed Materials.

   *a*) **Host License –** Subject to the terms of this Agreement, TI hereby grants to you a limited, non-exclusive license to use the Licensed Materials to generate applications that execute solely and exclusively on semiconductor devices manufactured by or for TI. Use of any software applications generated using the Licensed Materials on semiconductor devices manufactured by or for an entity other than TI is a material breach of this Agreement. It is also a material breach of this license to use the Licensed Materials to assist in the design, development or verification of a device that is instruction set compatible with, or functionally equivalent to, a TI semiconductor device.

   *b*) **Target License –** The Licensed Materials may include TI proprietary software programs in an object code format that are designed specifically for use in target applications. TI hereby grants to you a limited, non-exclusive license to

reproduce and distribute an unlimited number of copies of such programs solely and exclusively for use with semiconductor devices manufactured by or for TI. The Licensed Materials may also include libraries or applications software in a source code format. You may use these programs to create modified or derivative programs that may be reproduced and distributed by you provided that such programs execute solely and exclusively on semiconductor devices manufactured by or for TI and provided further that such programs are distributed only in an object code or executable format. You may not distribute, publish, rent, lease or sub-license the Licensed Materials, or any portion or derivative thereof, in a source code format or transfer or assign this Agreement without TI's prior written permission.

*c*) **Termination –** This license is effective until terminated. Without prejudice to any other rights, TI may terminate your right to use the Licensed Materials and any applications generated using the Licensed Materials under this Agreement if you fail to comply with the terms of this Agreement. In such event, you shall destroy all copies of the Licensed Materials, including all portions and derivatives thereof, in your possession, custody or control.

2. **Intellectual Property Rights –** The Licensed Materials contain copyrighted material, trade secrets and other proprietary information. In order to protect the Licensed Materials, and except as specifically permitted by statute by a provision that cannot be waived by contract, you may not unlock, decompile, reverse engineer, disassemble or otherwise translate any binary or object code versions of the software programs included in the Licensed Materials to human-perceivable form. You also agree that you will use your best efforts to prevent your employees and contractors from unlocking, decompiling, reverse engineering, disassembling, modifying or translating the Licensed Materials. In no event may you alter, remove or destroy any copyright notice included in the Licensed Materials. TI reserves all rights not specifically granted under this Agreement.

3. **Upgrades, Updates and Plug-ins –** If the Licensed Materials are labeled as an upgrade, update or plug-in you must be properly licensed to use the product identified by TI as being eligible for such upgrade, update or plug-in to use the Licensed Materials. An upgrade, update, or plug-in replaces or supplements a previously licensed eligible product. You may use the resulting upgraded product only in accordance with the terms of this Agreement and only to generate applications for use with semiconductor devices manufactured by or for TI. Notwithstanding the foregoing, nothing in the Agreement will be construed as an obligation for TI to maintain or support the Licensed Materials or to provide upgrades, updates or plug- ins to the

Licensed Materials.

4. **Warranties and Limitations –** YOU ACKNOWLEDGE AND AGREE THAT THE LICENSED MATERIALS ARE NOT INTENDED FOR PRODUCTION APPLICATIONS AND MAY CONTAIN IRREGULARITIES AND DEFECTS NOT FOUND IN PRODUCTION SOFTWARE. FURTHERMORE, YOU ACKNOWLEDGE AND AGREE THAT THE LICENSED MATERIALS HAVE NOT BEEN TESTED OR CERTIFIED BY ANY GOVERNMENT AGENCY OR INDUSTRY REGULATORY ORGANIZATION OR ANY OTHER THIRD PARTY ORGANIZATION. YOU AGREE THAT PRIOR TO USING, INCORPORATING OR DISTRIBUTING THE LICENSED MATERIALS IN ANY COMMERCIAL PRODUCT THAT YOU WILL THOROUGHLY TEST THE PRODUCT AND THE FUNCTIONALITY OF THE LICENSED MATERIALS IN THAT PRODUCT AND BE SOLELY RESPONSIBLE FOR ANY PROBLEMS OR FAILURES. THE LICENSED MATERIALS ARE PROVIDED "AS IS". TI MAKES NO WARRANTIES OR REPRESENTATIONS, EXPRESS, IMPLIED OR STATUTORY, INCLUDING ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, LACK OF VIRUSES, ACCURACY OR COMPLETENESS OF RESPONSES, RESULTS AND LACK OF NEGLIGENCE. TI DISCLAIMS ANY WARRANTY OF TITLE, QUIET ENJOYMENT, QUIET POSSESSION, AND NON- INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS WITH REGARD TO THE LICENSED MATERIALS OR YOUR USE OF THOSE MATERIALS. IN NO EVENT SHALL TI, OR ANY APPLICABLE LICENSOR, BE LIABLE FOR ANY SPECIAL, INDIRECT, INCIDENTAL, OR PUNITIVE DAMAGES, HOWEVER CAUSED, ARISING IN ANY WAY OUT OF THIS AGREEMENT, OR YOUR USE OF THE LICENSED MATERIALS, WHETHER OR NOT TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THESE EXCLUDED DAMAGES INCLUDE, BUT ARE NOT LIMITED TO, COST OF REMOVAL OR REINSTALLATION, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, OUTSIDE COMPUTER TIME, LABOR COSTS, LOSS OF DATA, LOSS OF GOODWILL, LOSS OF PROFITS, LOSS OF SAVINGS, LOSS OF USE OR INTERRUPTION OF BUSINESS OR OTHER ECONOMIC LOSS. IN NO EVENT WILL TI'S AGGREGATE LIABILITY UNDER THIS AGREEMENT OR ARISING OUT OF YOUR USE OF THE LICENSED MATERIALS EXCEED FIVE HUNDRED U.S. DOLLARS (U.S. $500). Because some jurisdictions do not allow the exclusion or limitation of incidental or consequential damages or limitation on how long an implied warranty lasts, the above limitations or exclusions may not apply to you.

5. **Export Control –** You hereby acknowledge and agree that unless prior authorization

is obtained from the United States Department of Commerce, neither you nor your customers will export, re-export, or release, directly or indirectly, any technology, software, or software source code (as defined in Part 772 of the Export Administration Regulations of the U.S. Department of Commerce ("EAR")), received from TI, or export, directly or indirectly, any direct product of such technology, software, or software source code (as defined in Part 734 of the EAR), to any destination or country to which the export, re-export or release of the technology, software, software source code, or direct product is prohibited by the EAR. The assurances provided for herein are furnished to TI by you in compliance with Part 740 (Technology and Software Under Restriction) of the EAR.

6. **Governing Law, Jurisdiction and Severability -** This Agreement will be governed by and interpreted in accordance with the laws of the State of Texas, without reference to that state's conflict-of-laws principles. This Agreement shall not be governed by the United Nations Convention on Contracts for the International Sale of Goods, nor shall it be governed by the Uniform Computer Information Transactions Act. Any dispute arising out of or related to this Agreement will be brought in, and each party consents to exclusive jurisdiction and venue in, the state and federal courts sitting in Dallas County, Texas. Each party waives all defenses of lack of personal jurisdiction and forum non-conveniens and agrees that process may be served on either party in a manner authorized by applicable law or court rule. If for any reason a court of competent jurisdiction finds any provision of the Agreement to be unenforceable, that provision will be enforced to the maximum extent possible to effectuate the intent of the parties and the remainder of the Agreement shall continue in full force and effect.

7. **Entire Agreement -** This is the entire Agreement between you and TI and supercedes any prior agreement between the parties related to the subject matter of this Agreement. No amendment or modification of this Agreement will be effective unless in writing and signed by a duly authorized representative of TI. You hereby warrant and represent that you have obtained all authorizations and other applicable consents required empowering you to enter into this Agreement.